НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“ ” 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки ***6.050103 «Програмна інженерія»***

на тему *Веб-застосування пошуку менторів та помічників з програмування*

Виконав: студент IV курсу, групи *ІП-51 Волков Ілля Андрійович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Рецензент:

_____
посада,науковий ступінь,вчене звання,прізвище,ініціали (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – *6.050103 «Програмна інженерія» (Програмне забезпечення систем)*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *О.А. Павлов*
(підпис) (ініціали, прізвище)
“___”_____2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Волкову Іллі Андрійовичу
(прізвище, ім`я, по батькові)

1. Тема проекту «*Веб-застосування пошуку менторів та помічників з програмування*»
керівник проекту *Ліщук Катерина Ігорівна, к.т.н., доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “23” квітня 2019 р. №1181-с
2. Термін подання студентом проекту «03» *червня 2019 року*

3. Вихідні дані до проекту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
3) Розгортання та впровадження програмного забезпечення
4) Керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1) Схема бази даних

2) Схема структурна варіантів використання

3) Схема структурна бізнес процесу

4) Креслення вигляду екранних форм

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «12» *березня 2018 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	*Вивчення рекомендованої літератури*	*20.03.2019*	
2.	*Аналіз існуючих методів розв'язання задачі*	*01.04.2019*	
3.	*Постановка та формалізація задачі*	*10.04.2019*	
4.	*Аналіз вимог до програмного забезпечення*	*25.04.2019*	
5.	*Алгоритмізація задачі*	*10.05.2019*	
6.	*Моделювання програмного забезпечення*	*12.05.2019*	
7.	*Обґрунтування використовуваних технічних засобів*	*12.05.2019*	
8.	*Розробка архітектури програмного забезпечення*	*15.05.2019*	
9.	*Розробка програмного забезпечення*	*23.05.2019*	
10.	*Налагодження програми*	*25.05.2019*	
11.	*Виконання графічних документів*	*28.05.2019*	
12.	*Оформлення пояснювальної записки*	*28.05.2019*	
13.	*Подання ДП на попередній захист*	*30.05.2019*	
14.	*Подання ДП рецензенту*	*06.06.2019*	
15.	*Подання ДП на основний захист*	*08.06.2019*	

Студент _____ Волков І.А.
(підпис)

Керівник проекту _____ Ліщук К.І.
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-5102.045440.02.81	Пояснювальна записка	78	
3	А4	КПІ.ІП-5102.045440.03.91	Технічне завдання	11	
4	А4	КПІ.ІП-5102.045440.04.13	Опис програми	15	
5	А4	КПІ.ІП-5102.045440.05.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-5102.045440.06.34	Керівництво користувача	7	
7	А3	КПІ.ІП-5102.045440.07.99.СС	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-5102.045440.07.99.СС	Схема структурна бізнес процесу	1	
9	А3	КПІ.ІП-5102.045440.07.99.СБД	Схема бази даних	1	
10	А3	КПІ.ІП-5102.045440.07.99.КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-5102.045440.01.90		
Розробн.		Волков І.А.			Відомість дипломного проекту	Літ.	Лист 1 / Листів 1
Керівн.		Ліщук К.І.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-51	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка до дипломної роботи: 122 сторінки, 21 рисунок, 30 таблиць, 9 посилань.

Об’єкт дослідження: задача пошуку менторів та помічників з програмування.

Мета роботи: спрощення процесу знаходження професійного ментора для тренування конкретних навичок практикантів шляхом автоматизації і оптимізації процесів пошуку та комунікації між практикантами та менторами, та заохочення менторів до навчання практикантів.

В першому розділі проведено аналіз вимог до програмного забезпечення, наведені функціональні та нефункціональні вимоги, наведено постановку завдання.

В другому розділі виконане моделювання програмного забезпечення, наведені його архітектура та подробиці реалізації.

В третьому розділі розроблені процеси тестування даного програмного забезпечення, а також проведений його аналіз якості.

В четвертому розділі розроблені процеси розгортання програмного забезпечення, та наведена інструкція користувача.

ВЕБ-ЗАСТОСУВАННЯ. ПОШУК МЕНТОРІВ, ОПТИМІЗАЦІЯ ПРОЦЕСУ ПОШУКУ, ЗАОХОЧЕННЯ МЕНТОРІВ, КОМУНІКАЦІЯ МІЖ МЕНТОРАМИ ТА ПРАКТИКАНТАМИ.

ANNOTATION

Explanatory note to the thesis: 122 pages, 21 figures, 30 tables, 9 links.

Object of research: the task of finding mentors and programming assistants.

Purpose: simplifying of the process of finding a professional mentor to train specific skills of practitioners by automating and optimizing the search and communication processes between practitioners and mentors, and encouraging mentors to train apprentices.

In the first section, the analysis of software requirements was performed, functional and non-functional requirements were given, the task set out.

In the second section, software modeling is performed, its architecture and implementation details are given.

In the third section, the processes for testing this software are developed, as well as its quality analysis.

In the fourth section, the software deployment processes are developed, and the user's guide is provided.

WEB-APPLICATION. SEARCH OF MENTORS, OPTIMIZATION OF THE SEARCH PROCESS, MENTORING, COMMUNICATION BETWEEN MENTORS AND PRACTICES.

Пояснювальна записка
до дипломного проекту

на тему: ВЕБ-ЗАСТОСУВАННЯ ПОШУКУ МЕНТОРІВ ТА ПОМІЧНИКІВ З ПРОГРАМУВАННЯ

Київ – 2019 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Менторинг (або більш звична назва - наставництво) – це система відносин двох (інколи більше) людей, коли людина, що має більший досвід у деякій сфері людської діяльності, передає свої знання та навички людям з меншим досвідом. Як правило, такі відносини відбуваються шляхом виконання цими людьми певної діяльності під керівництвом більш досвідченої особи.

Django - вільний фреймворк для веб-додатків на мові Python, що використовує шаблон проектування MVC [1)].

Django REST Framework - бібліотека, яка працює зі стандартними моделями Django для створення гнучкого і потужного API для проекту [2)].

Angular - JavaScript-фреймворк з відкритим вихідним кодом. Призначений для розробки односторінкових додатків. Його мета - розширення браузерних додатків на основі MVC-шаблону, а також спрощення тестування і розробки [3)].

MVC (Model View Controller) - схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно [4)].

Dependency injection - це метод, за допомогою якого один об'єкт (або статичний метод) постачає залежності іншого об'єкта. Залежність - це об'єкт, який можна використовувати (служба). Ін'єкція - це передача залежності на залежний об'єкт (клієнт), який буде використовувати його [5)].

ВСТУП

У сучасному світі дуже гостро постає питання підготовки молодих спеціалістів до роботи на підприємствах у реальних умовах, а не на лавах університету. Досить часто університет не в змозі надати необхідні знання для того щоб студент міг одразу з лави університету стати реально корисним співробітником на виробництві чи в офісі. І це стосується навіть не якогось конкретного університету, а системи вищої освіти в цілому. Колись університети готували спеціалістів не за «престижем» спеціальності, а за державним замовленням, де саме держава вирішувала, де і які спеціалісти їй потрібні. Тому і робота була для всіх спеціальностей однаково доступна, і не було таких проблем з безробіттям та з еміграцією кваліфікованих кадрів за кордон. За нинішніх умов високої конкуренції на кожне пристойне робоче місце одних університетських знань замало. Для роботодавця ти повинен чимось виділятися із «сірого» натовпу випускників. В першу чергу це стосується сфери розробки та супроводу програмного забезпечення. Це зумовлено надзвичайно швидкими темпами росту даної індустрії, та кількістю знань та навичок, що необхідні для початку роботи навіть над найпростішими реальними розробками. Жодна навчальна програма не містить такого обсягу матеріалу. Але існує проблема також і із існуючим навчальним матеріалом. Навіть найсучасніша університетська програма відстає від реального технологічного стеку сучасних ІТ-компаній на декілька років. Інша причина – це необхідність у сфері ІТ для реальної роботи низки практичних навичок, які дуже сильно відрізняються у різних галузях цієї індустрії, більшість з яких університетська програма охопити просто не в змозі. Та і навички у програмуванні надаються нею дуже і дуже базові. Принцип «всього по трохи» працює лише як ознайомчий матеріал, придатний лише для вибору якоїсь цікавої для студента теми та подальшого її самостійного вивчення.

Самоосвіта – ось чим вимушені займатись студенти до того як потрапити на свою першу роботу в ІТ. Існує декілька основних підходів до неї.

Онлайн - курси є досить непоганим рішенням, адже вони є як за невеликі гроші, так і цілком безкоштовні. Основним їх недоліком є майже повна відсутність зворотного зв’язку. Тобто, дуже важко, а, інколи, і зовсім неможливо задати питання чи попросити поради у викладача. А спілкування є основою будь-якого педагогічного процесу. Такий недолік часто призводить до засвоєння помилок та до появи прогалин у знаннях, які не так просто заповнити.

Іншим підходом є очні курси, на які зазвичай ходять досить великими групами і цей недолік проявлений менше. Але, на відміну від попереднього підходу, очні курси вимагають набагато більше вільного часу та не дозволяють налаштувати графік занять під себе. І коштують набагато дорожче зазвичай.

Одним з найефективніших рішень даної проблеми є процес так званого менторингу, або, іншими словами, наставництво. Цей процес призначений для передачі практичних навичок та знань від більш досвідченого співробітника до менш досвідченого шляхом співпраці віч-на-віч або у невеликих групах в процесі вирішення реальних задач з розробки або супроводу програмного забезпечення. Він має свої недоліки, які будуть розглянуті в рамках цієї дипломної роботи та будуть знайдені і реалізовані шляхи для вирішення деяких з цих проблем.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Менторинг – це вид професійної діяльності, коли людина з більшим досвідом передає свої знання та навички в певній сфері людської діяльності людині з меншим досвідом.

Сучасний термін менторинг часто плутають із схожим по змісту наставництвом. На відміну від наставництва, менторинг ставить перед собою задачу не стільки навчити людину навичкам і опанувати знання які є у вільному доступі, скільки дати людині неочевидні знання та навички, які приходять тільки з досвідом реальних розробок програмного забезпечення.

Під час менторингу надається перевага великій кількості практичних занять і самостійне засвоєння теоретичного матеріалу (повна протилежність університетській програмі).

Менторинг поділяють за кількістю людей, що беруть у ньому участь, на індивідуальний, груповий та колективний.

1.2 Змістовний опис і аналіз предметної області

Визначальною особливістю менторингу є індивідуальний підхід до кожного практиканта. Тобто, ментор окремо визначає рівень підготовки кожного учня, підбирає під нього навчальний план, можливо назначає йому додаткові заняття (у випадку групового менторингу), займається мотивацією своїх учнів, та оцінює їх успіхи відносно початкового рівня знань. Найчастіше таким видом наставництва займаються компанії для підготовки власних спеціалістів для вирішення своїх специфічних задач. Адже наймати готового спеціаліста і дорожче, і, інколи, не має сенсу, бо всіх необхідно навчати в той чи іншій мірі.

Практика проведення такого менторингу відрізняється у різних випадках в залежності від поставленої задачі.

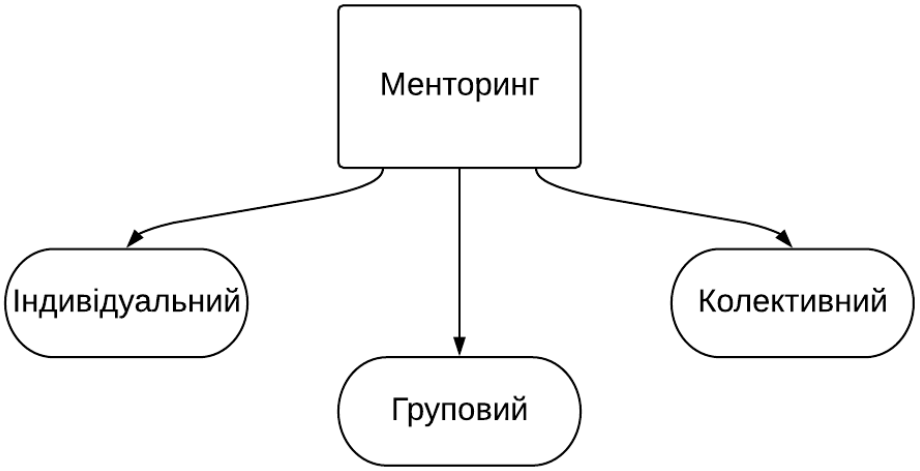


Рисунок 1.1 - Види менторингу

Найбільш ефективним видом менторингу є індивідуальний. В такому випадку ментор займається зі своїм практикантом віч-на-віч. З одного боку, такий метод дозволяє практиканту найшвидше опанувати необхідні навички, з іншого – в цьому випадку його навчання виявляється найбільш ресурсоємним (витрачається найбільше часу програміста на одну людину). Використовується невеликими компаніями для штучної підготовки необхідних фахівців.

Протилежним підходом є менторинг груповий, під час якого один ментор навчає групу людей. Цей підхід є найбільш ефективним для великих компаній, яким необхідно масово готувати спеціалістів певного профілю. Він є значно вигіднішим, проте потребує в середньому більше часу на підготовку конкретного учня. Також практично нівелюється така перевага менторинга, як індивідуальний підхід.

Компромісом є колективний менторинг, коли декілька менторів навчають групу людей. Це дозволяє в певній мірі сумістити переваги обох попередніх підходів разом, проте далеко не на 100 відсотків. Також використовується здебільшого у великих компаніях.

Але, незалежно від виду менторингу, його ціль – це розвиток навичок та збагачення знань у деякій предметній області. Дуже добре цю ціль ілюструє модель навчання Д. Колба:

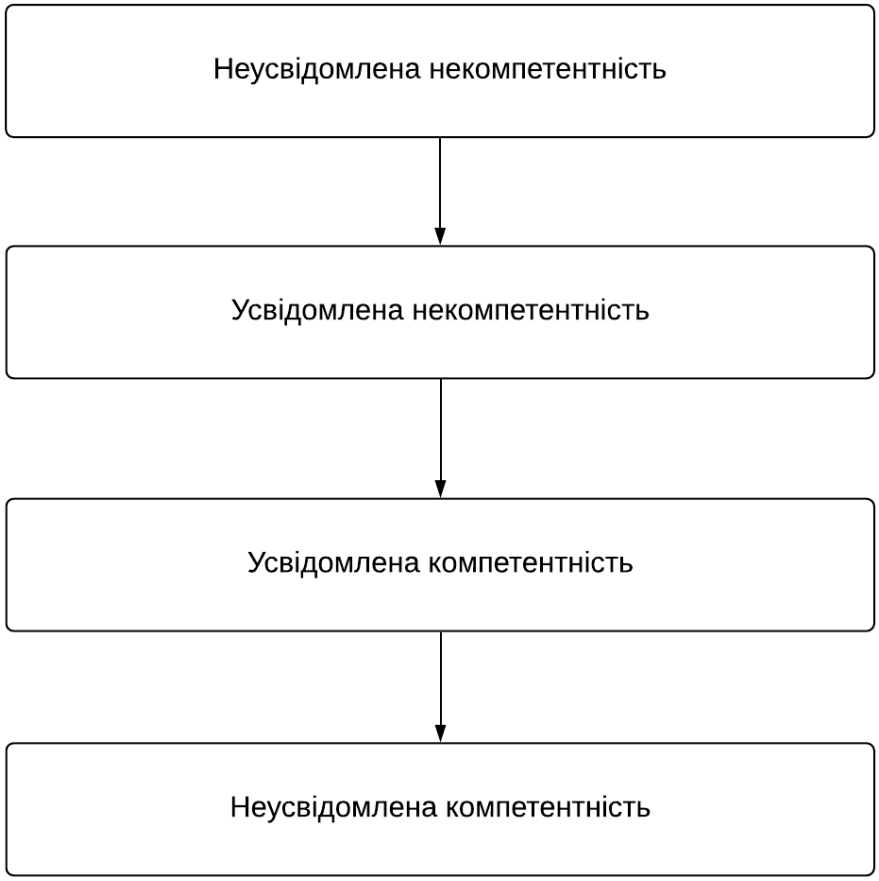


Рисунок 1.2 - Модель (цикл) навчання Д. Колба

Згідно цієї моделі, навички людини у будь-якій предметній області поділяються на чотири рівня:

- неусвідомлена некомпетентність (людина не усвідомлює що вона некомпетентна, а тому робить багато помилок);
- усвідомлена некомпетентність (людина вже усвідомлює що вона некомпетентна, починає ставити правильні запитання та розуміє, в якому напрямку вона має розвиватися);

– усвідомлена компетентність (людина компетентна, але застосування її знань на практиці вимагає чіткого усвідомлення та значних розумових зусиль);

– неусвідомлена компетентність (людина компетентна, і застосування її знань відбувається автоматично на рівні навичок).

Ціллю менторингу є підвищення рівня навичок практиканта в певній предметній області з першого рівня до четвертого.

Ця модель навчання дуже добре працює на практиці, проте має декілька суттєвих недоліків. По-перше, досить важко знайти необхідного спеціаліста для даного тренінгу. Звісно, більшість компаній надають таку допомогу безкоштовно після працевлаштування, але це і є проблемою: існують компанії, які для працевлаштування вимагають від кандидата навіть на посаду молодшого спеціаліста низки практичних навичок, які майже неможливо здобути самостійно. По-друге, іноді цих менторів майже змушують працювати із молодими співробітниками, що вкупі з їх посередніми педагогічними вміннями випливає в низку проблем в комунікації, і, як наслідок, в посередній результат навчання. Ну і по-третє, досить часто для здобуття необхідних навичок одного ментора замало, а компанії часто не бажають витрачати час інших розробників, навіть якщо це принесе прибуток у майбутньому.

Враховуючи все вище перелічене, досить багато випускників з IT-спеціальностей мають проблему із знаходженням свого першого місця роботи.

З користю менторингу для практикантів все більш-менш зрозуміло. Але це лише одна сторона медалі. Сам процес досить корисний і для самого ментора:

– індивідуальні заняття з практикантами дозволяють вдосконалити свої педагогічні та комунікативні здібності;

– заняття із практикантами що мають різний рівень підготовки дозволяють покращити та повторити як базові знання та навички, так і більш поглиблені;

– постійні «незручні» питання практикантів дозволяють заповнити більшість «білих плям» у знаннях ментора та поглибити знання у неочікуваних областях.

Але, як і у випадку з практикантами, ментори також мають певні проблеми.

а) Пошук практикантів. Далеко не всі компанії проводять постійну підготовку нових спеціалістів. Деякі, особливо невеликі, роблять це виключно за потребою під певний проект. Тут уже навіть не йдеться мова про навчання практиканта за знайомою технологією, а про навчання хоч якого-небудь практиканта.

б) Мотивація. Не всі ІТ-компанії проводять заохочення менторів, які бажають навчати своїх співробітників. Або не дають з огляду на те, що це займає багато робочого часу і не хочуть вкладатись у майбутнє. Або просто не дають людям, які захотіли спробувати себе ментором підтримки, аби здійснити це бажання.

Тож, не дивлячись на небувалий попит на такий вид навчання, він все ще має багато проблем та багато шляхів для покращення, які будуть розглянуть далі.

1.3 Аналіз успішних ІТ-проектів

На ринку вже існують декілька програмних рішень проблем менторингу. Це є досить популярні інтернет – ресурси, які пропонують своїм користувачам різного ступеня зручності системи пошуку менторів для практикантів і також різного ступеня зручності систем для реєстрації менторів. Далі будуть розглянуті дані рішення.

1.3.1 Аналіз відомих технічних рішень

Відомі технічні рішення з пошуку менторів поділяються на три основні типи:

– пошук менторів за категоріями;
– пошук менторів за ключовими словами;
– мішаний тип пошуку.

Пошук менторів за категоріями реалізується шляхом наявності деякої класифікації менторів, наприклад, за їх технічними навичками, і наданням користувачу окремих списків менторів згідно цієї класифікації. Перевагами такого підходу є зручність використання та наочність процесу, а основним недоліком – недостатня гнучкість (дуже складно знайти конкретного ментора якщо ти знаєш його ім'я, проте не знаєш, які технічними навички він має).

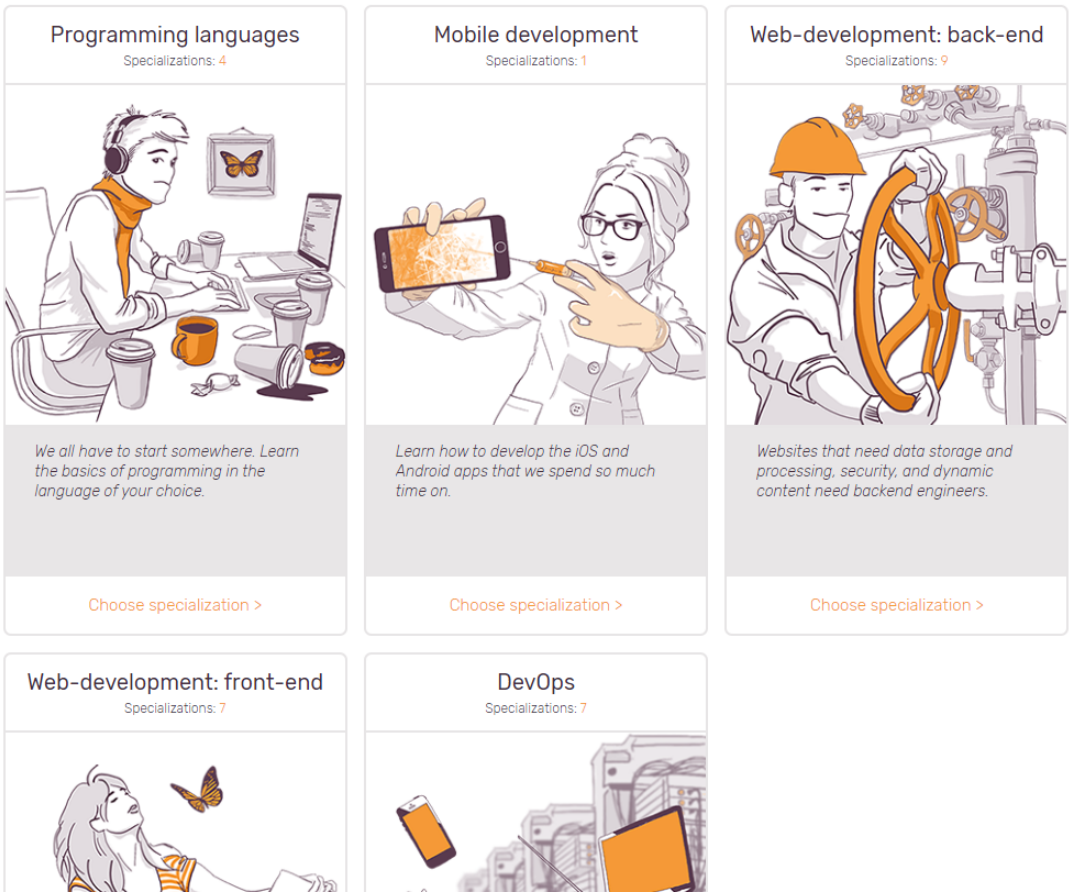


Рисунок 1.3 - Приклад системи пошуку за категоріями

Пошук менторів за ключовими словами реалізується шляхом наявності пошукової форми на екрані і відправки запиту на сервер, повертаючи на клієнт дані, відфільтровані згідно змісту цієї пошукової форми. Перевагами такого підходу є досить висока гнучкість (фільтрація у базі даних зі сторони сервера

може відбуватися одразу за декількома полями одночасно за результатами одного запиту) і простота реалізації. Проте основним недоліком є відсутність наочності процесу пошуку, адже користувач повинен здогадуватися сам, які поля у базі даних можуть брати участь у фільтрації результатів пошукового запиту.

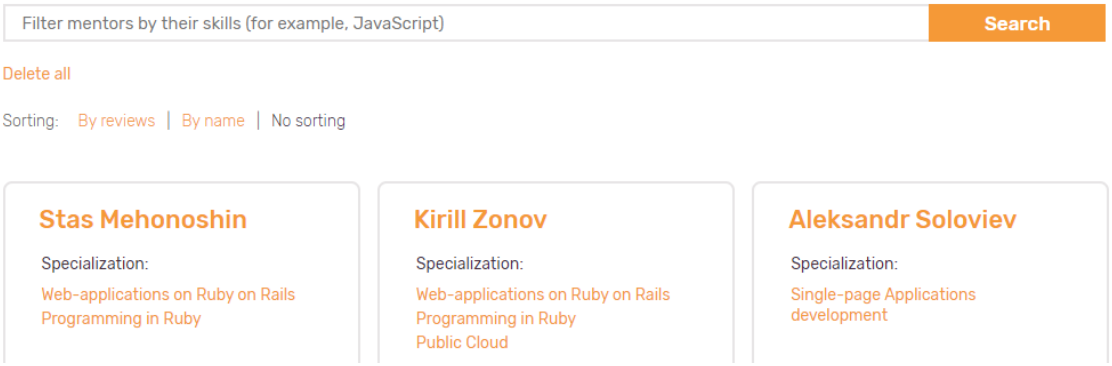


Рисунок 1.4 - Приклад системи пошуку за ключовими словами

Переваги та недоліки такого методу є протилежні до попереднього, тому досить часто їх намагаються об’єднати в одному проекті. Для цього існують два основних підходи.

Перший полягає у простому об’єднанні в одному проекті цих двох методів пошуку, як повністю незалежних один від одного. Цей підхід є дещо простішим у реалізації, проте не позбавлений недоліків попередніх. Він лише дозволяє користувачеві обирати між перевагами та недоліками перших двох методів пошуку. Але він все ж не дає всіх переваг одночасно. Також такий підхід досить складний у реалізації, адже вона, на відміну від попередніх, вимагає вдвічі більше часу на розробку та тестування.

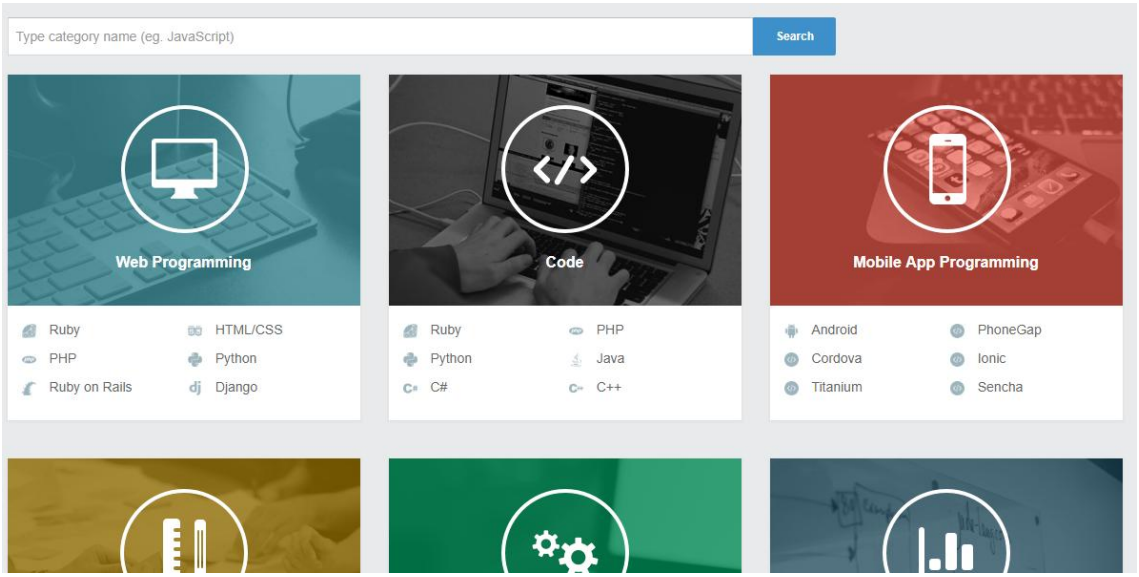


Рисунок 1.5 - Приклад об'єднання попередніх методів, як незалежних один від одного

Останнім і, на мою думку, найбільш оптимальним рішенням є суміщення перших двох підходів наступним чином: система дозволяє одночасну фільтрацію даних як категоріями, так і за ключовими словами в рамках одного пошукового запиту. Цей підхід дещо складніший в реалізації, проте поєднує в собі всі переваги попередніх підходів та позбавлений від їх недоліків. У випадку цього дипломного проекту буде обраний саме він.

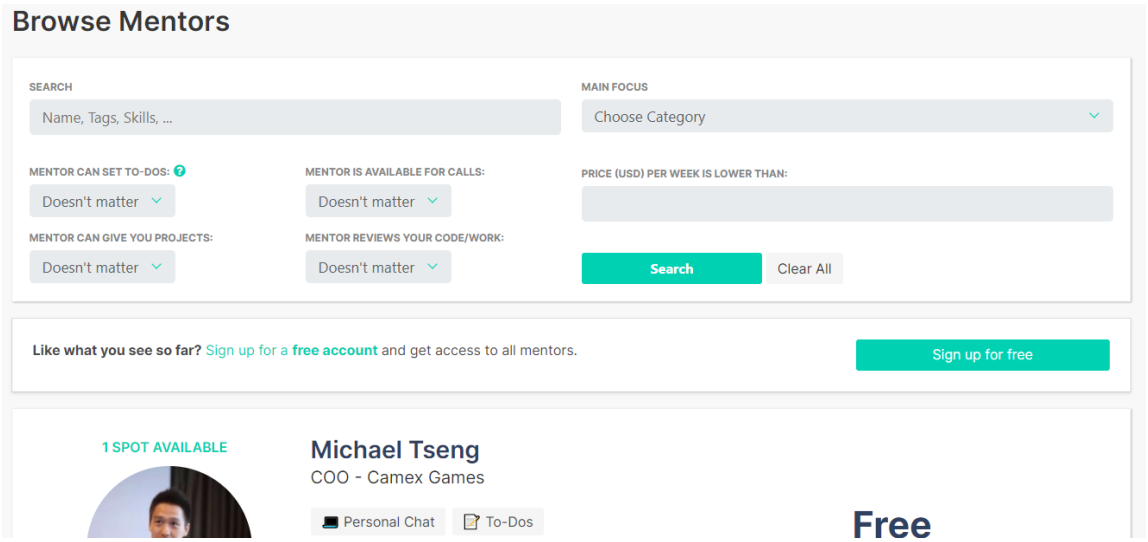


Рисунок 1.6 - Приклад об'єднання попередніх методів в рамках одного пошукового запиту

1.3.2 Аналіз відомих програмних продуктів

а) Mkdev (https://mkdev.me) – веб-застосування для пошуку менторів з програмування. На даному ресурсі можна наймати наставників (у вигляді підписки на певний період часу). Підписка включає обмін повідомленнями в приватному чаті з ментором, індивідуальний навчальний план, практичні завдання та огляд коду.

З переваг слід виокремити лаконічність інтерфейсу користувача та завершену і повноцінну систему з пошуку та навчання у ментора. З недоліків – відсутність системи пошуку менторів за допомогою запитів та відсутність рейтингової системи менторів (можна лише почитати відгуки, проте вони, найчастіше, не в змозі дати повну картину навичок викладача).

Пошук менторів реалізований окремо за ключовими словами і за категоріями. Реєстрація кандидатури користувача – ментора відбувається за допомогою анкети – опитування.

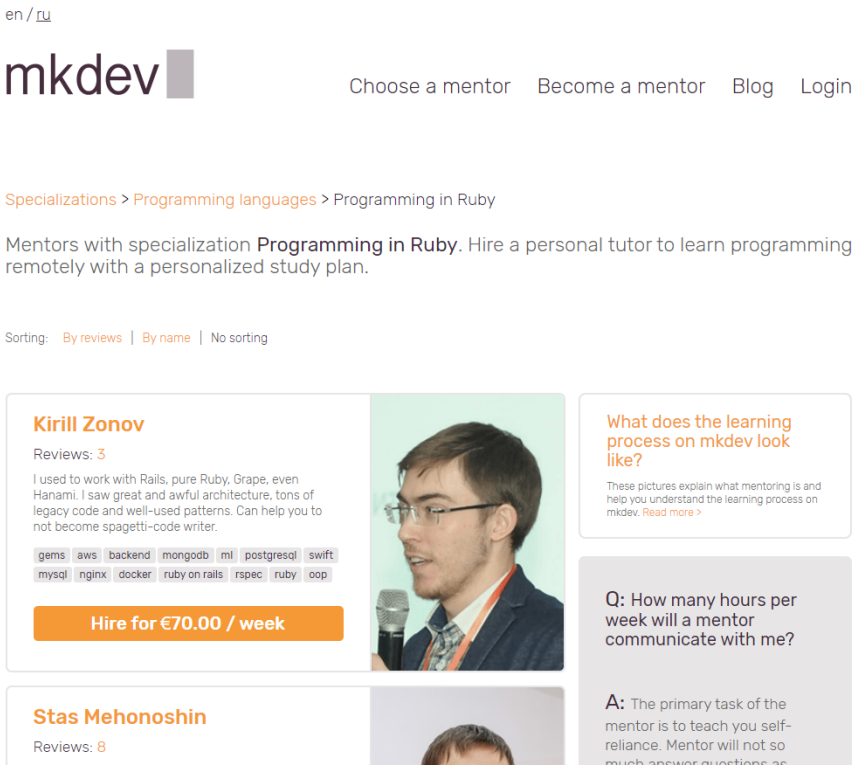


Рисунок 1.7 - Скріншот з ресурсу https://mkdev.me

б) Codementor (https://www.codementor.io/) – веб-застосування, що виконує ту ж саму функцію, як і попереднє, але має більш вдало проробленний інтерфейс, який вміщає в себе більше інформації. Навчання у наставника відбувається також у вигляді платної підписки.

В даному проекті реалізована система пошуку менторів за ключовими словами. Реєстрація ментора відбувається за допомогою окремої логін – форми.

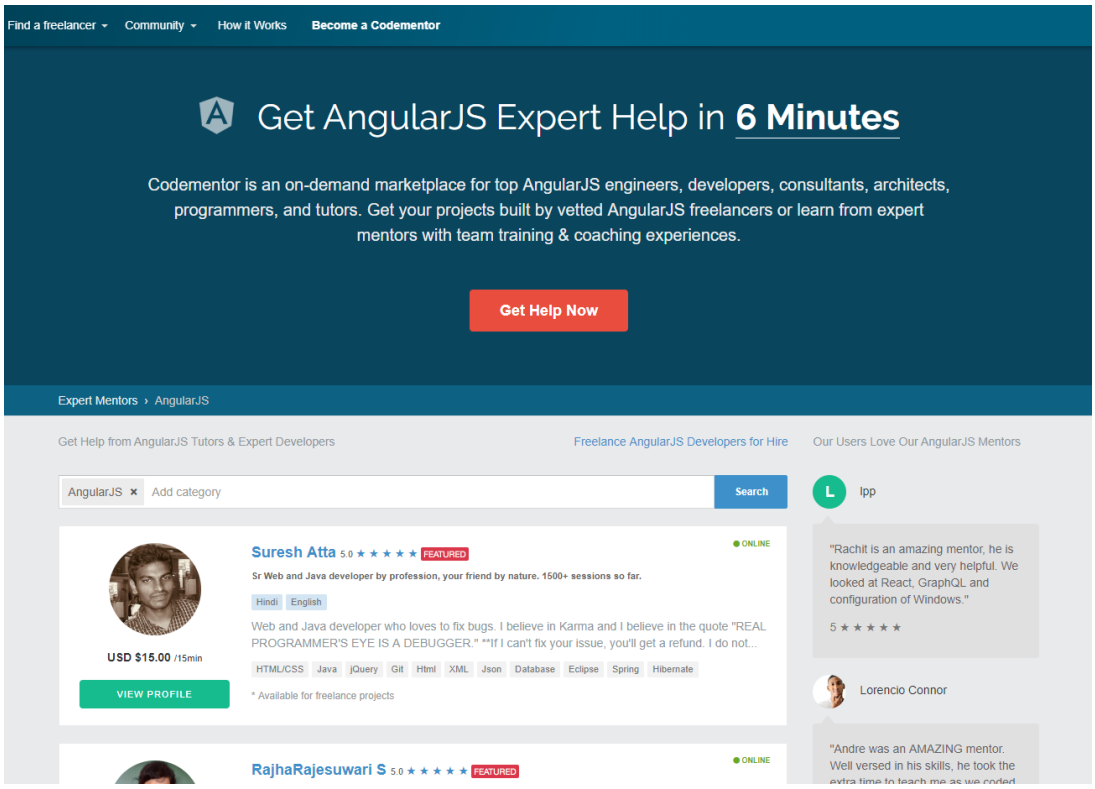


Рисунок 1.8 - Скріншот з ресурсу https://codementor.io

в) Mentorcruise (https://mentorcruise.com/) – веб-застосування, що є аналогом попереднього. Має схожий функціонал, але процес підбору менторів є більш гнучким і зручним, ніж у попередньому рішенні. Найбільш вдало реалізоване рішення задачі пошуку менторів, на мій погляд.

Реєстрація ментора відбувається шляхом заповнення детальної анкети, що надає найбільш розгорнуту інформацію іншим користувачам.

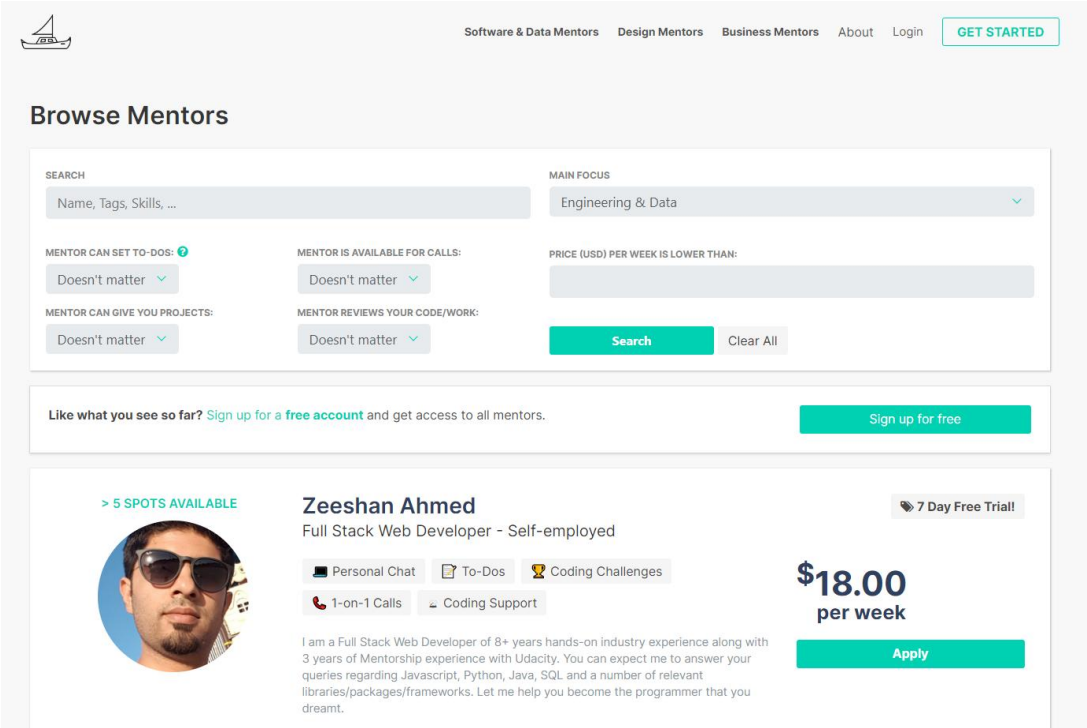


Рисунок 1.9 - Скріншот з ресурсу https://mentorcruise.com

1.4 Аналіз вимог до програмного забезпечення

1.4.1 Розроблення функціональних вимог

Основними дійовими особами в рамках даного проекту є практиканти і ментори. Об’єктами, над якими виконуються основні операції є каталог замовлень та заявки на виконання замовлень.

Неавторизовані користувачі – це будь-які відвідувачі веб-ресурсу до реєстрації та входу в систему.

Практиканти – це звичайні користувачі веб-застосування, що виконують функції пошуку менторів та створюють замовлення на наставництво у деякій заданій предметній області. Також вони виконують оцінювання персональних навичок ментора та можуть виступати ініціаторами спілкування з ними.

Ментори – це користувачі веб-застосування, що мають можливість приймати замовлення у практикантів на навчання. Мають певний список навичок, за якими вони можуть навчати практикантів. Мають систему

оцінювання, згідно якої вони відображаються у пошуку та контактну інформацію, за допомогою якої практиканти можуть зв’язуватися з ментором.

Каталог замовлень – список замовлень на проведення менторингу. Включає в себе інформацію про практикантів, які бажають навчатися та про технології, які вони бажають вивчати.

Заявки на виконання замовлень – списки заявок, прикріплених до конкретних замовлень. Містять інформацію про ментора, який погоджується навчати практиканта за його вимогами.

Основними функціями даного веб – застосування є обмін повідомленнями, створення замовлень, створення заявок на виконання замовлень, прийняття заявок на виконання замовлень, виконання замовлень та оцінювання навичок менторів.

Обмін повідомленнями – функція обміну текстовими повідомленнями між менторами та практикантами. Є основним засобом комунікації в даному проекті.

Створення замовлень – функція створення замовлень на проведення менторингу. Створюються практикантами та додаються в каталог замовлень. Є ключовою функцією.

Створення заявок на виконання замовлень - функція створення заявок на виконання замовлень. Створюються менторами та додаються в каталог заявок на виконання конкретного замовлення. Є ключовою функцією.

Прийняття заявок на виконання замовлень - функція прийняття заявок на виконання замовлень. Приймаються практикантами, після чого замовлення вилучається з каталогу замовлення. Також на цьому етапі до замовлення прив’язується конкретний ментор. Є ключовою функцією.

Виконання замовлення - функція виконання замовлень. Рішення про виконання приймається ментором, після чого замовлення видаляється. Є ключовою функцією.

Оцінювання навичок менторів – функція оцінювання конкретних навичок менторів за допомогою 10-бальної оцінки та текстових коментарів. Також можна оцінювати загальну роботу ментора. Слугує для оцінювання педагогічних та професійних навичок ментора.

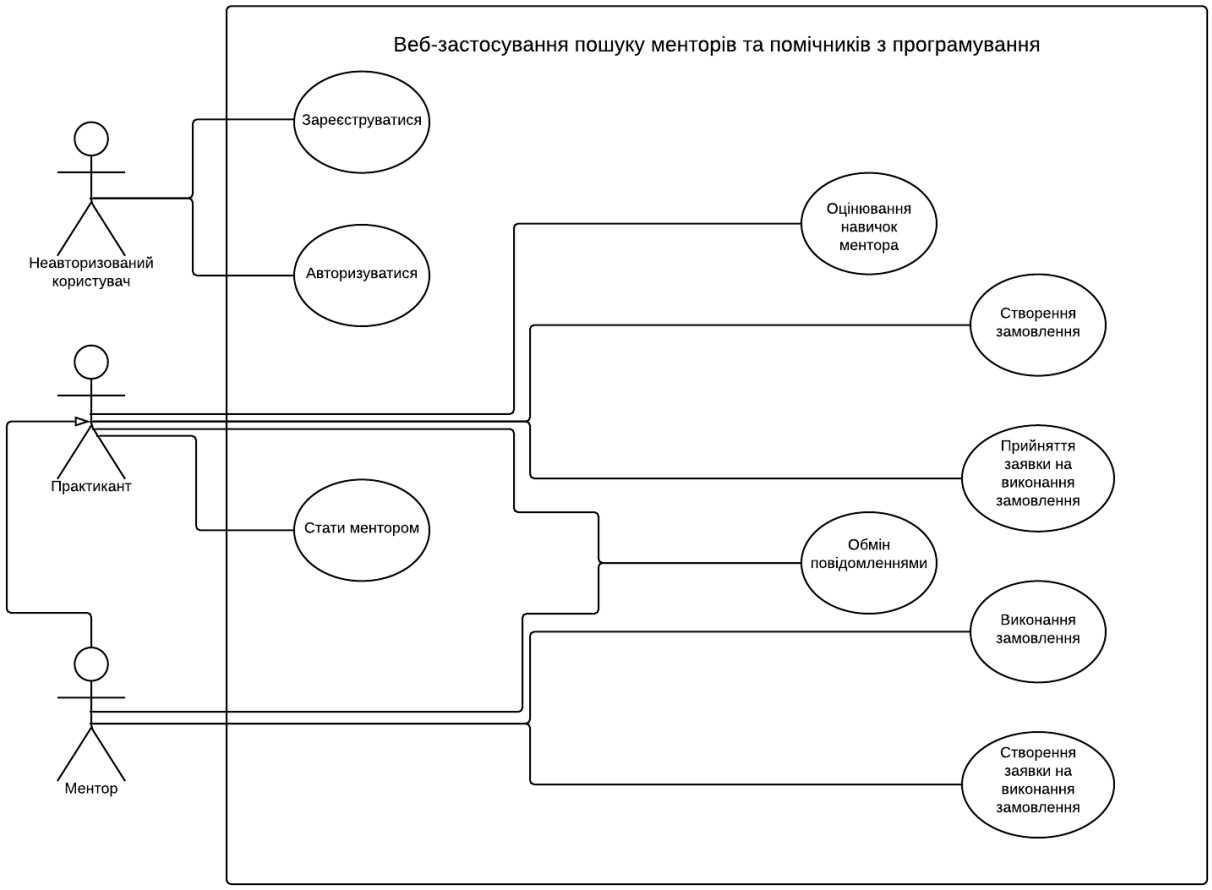


Рисунок 1.10 - Модель варіантів використання

Таблиця 1.1 - Функціональні вимоги

№	Назва функції	Код	Призначення функції	Актор	Пріоритет
1.	Авторизуватися	REQ1	Авторизація користувачів у системі	Неавторизований користувач	головний
2.	Зареєструватися	REQ2	Реєстрація користувачів у системі	Неавторизований користувач	головний
3.	Стати ментором	REQ3	Зміна ролі практиканта на ментора	практикант	головний
4.	Обмін повідомленнями	REQ4	Обмін текстовими повідомленнями	практикант, ментор	другорядний
5.	Створення замовлень	REQ5	Створення замовлення на проходження менторингу	практикант	головний
6.	Створення заявок на виконання замовлень	REQ6	Створення заявки на надання послуги менторингу	ментор	головний
7.	Прийняття заявок на виконання замовлень	REQ7	Прийняття заявки на надання послуги менторингу	практикант	головний

Продовження таблиці 1.1

8.	Виконання замовлення	REQ8	Відмітка про те що замовлення виконано у повному обсязі	ментор	головний
9.	Оцінювання навичок менторів	REQ9	Оцінювання роботи ментора за 10-бальною шкалою	практикант	другорядний

Таблиця 1.2 – Варіанти використання

№	Варіант використання	Код	Опис
1.	Пошук менторів	UC1	Пошук менторів користувачами-практикантами для подальшого спілкування
2.	Спілкування з менторами	UC2	Задання питань ментору за допомогою текстових повідомлень чи через його контактні дані
3.	Менторство	UC3	Дистанційне навчання практикантів
4.	Навчання	UC4	Дистанційне навчання у менторів
5.	Оцінювання менторів	UC5	Оцінювання навичок менторів

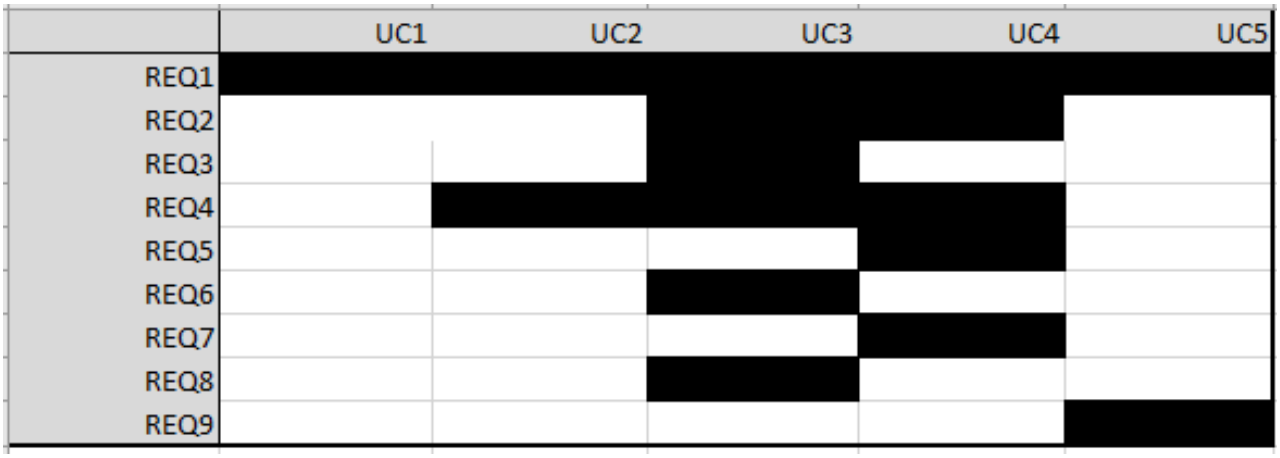


Рисунок 1.11 - Матриця залежності між функціональними вимогами та варіантами використання

1.4.2 Розроблення нефункціональних вимог

Основними нефункціональними вимогами для даного веб-застосунку є такі, що забезпечують найкращий досвід використання програми користувачами, а саме:

- інтуїтивний візуальний інтерфейс;
- висока стійкість до відмов;
- висока швидкість завантаження та видачі результатів запитів;
- наочне відображення помилок для користувача.

1.4.3 Постановка комплексу завдань модулю

1.4.3.1 Призначення розробки

Дана розробка призначена для пошуку менторів з різних технологій програмування, та налагодження співпраці між ментором та практикантом на взаємовигідних умовах. Також вона надає можливість будь-якому спеціалісту навчати та ділитись своїми знаннями і навичками з іншими спеціалістами. Розробка включає в себе систему оцінювання менторів, яка сприятиме успішності найбільш професійних фахівців.

1.4.3.2 Цілі та задачі розробки

Метою даної розробки є спрощення процесу знаходження професійного ментора для тренування конкретних навичок практикантів шляхом автоматизації і оптимізації процесів пошуку та комунікації між практикантами та менторами, та заохочення менторів до навчання практикантів. Також дана розробка надає можливість молодим спеціалістам здобути та перейняти у менторів базові практичні навички, що надзвичайно необхідні для виходу на ринок ІТ-спеціалістів. З боку менторів, дана розробка надає додаткові можливості для самореалізації висококваліфікованих спеціалістів та свого викладацького потенціалу.

Задачею розробки є створення веб-застосування для пошуку менторів з програмування, яке б містило в собі:

- систему пошуку менторів;
- систему оцінювання менторів;
- систему спілкування з менторами;
- систему навчання у менторів.

1.5 Висновки по розділу

Виходячи з усього вище описаного, можна дійти висновку, що предметна область даної розробки потребує оптимізації та покращення. На ринку вже існують кілька продуктів, що вирішують проблеми області менторингу програмістів, проте їх використання носить досить обмежений характер на сьогоднішній день. З цього можна дійти висновку, що ще одна спроба вирішити ці проблеми не буде зайвою та може принести користь суспільству.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Розглянемо основні варіанти поведінки користувачів у системі та опишемо детально кожен з них.

Основним бізнес – процесом даного веб-застосування є навчання у ментора. Воно відбувається за наступним алгоритмом:

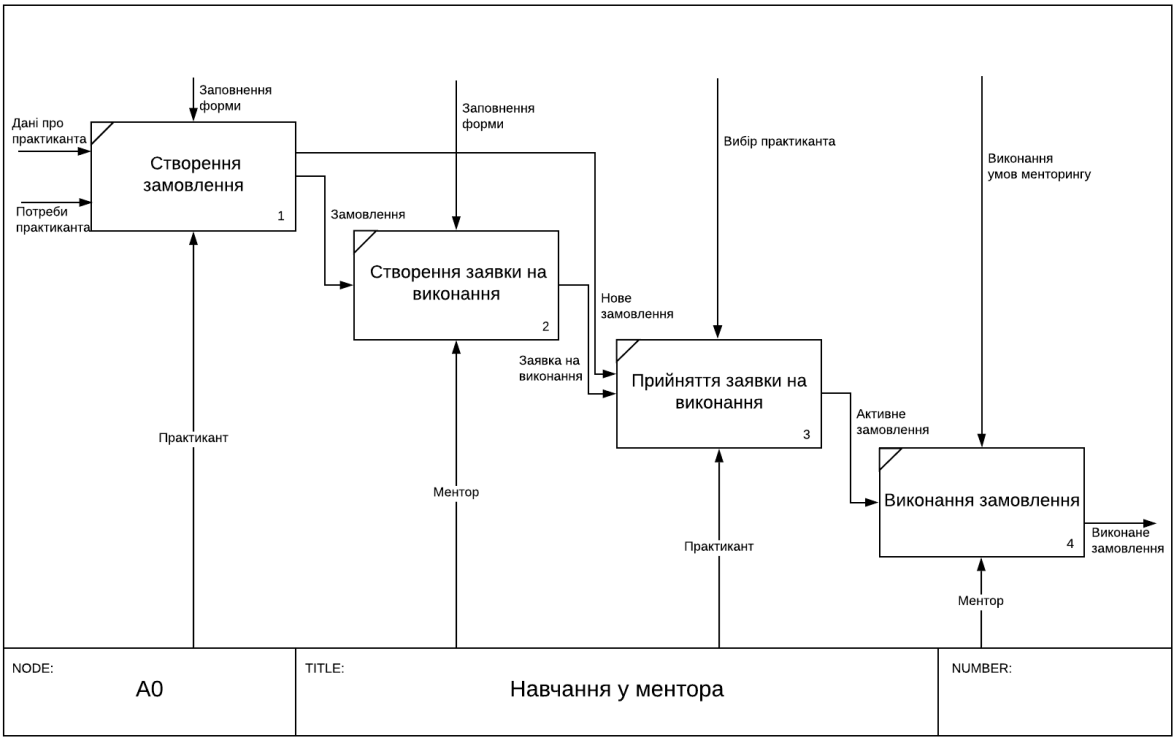


Рисунок 2.1 – Схема структурна бізнес-процесу процесу навчання у ментора

а) спочатку практикант самостійно заповнює на веб-сторінці форму, у якій він вказує свої побажання та технології, які він бажає вивчати. Після збереження ця форма стає доступною для перегляду всім менторам;

б) якщо ментора задовольняють умови контракту, він створює заявку на його виконання, автоматично погоджуючись на умови практиканта. Всі

заявки на виконання замовлення менторингу стають доступними для перегляду практикантом;

в) практикант з цих заявок обирає ментора, який найбільше йому до вподоби. Це дозволить як ментору навчати кого він хоче, так і практиканту навчатись в того в кого він хоче. Практикант приймає одну з заявок;

г) за допомогою обміну повідомленнями та контактами між ментором та практикантом, вони розпочинають своє спілкування та навчання. Коли ментор вирішить що навчання завершене, він завершує контракт;

д) практикант може оцінювати навички ментора по завершенню навчання, що вплине на його подальший рейтинг.

Коротко оглянемо менш суттєві процеси:

– реєстрація практиканта відбувається за допомогою звичайної реєстраційної форми;

– кожен практикант має змогу стати ментором, для цього йому потрібно заповнити додаткову форму з особистими даними;

– загальну оцінку ментора може поставити лише той практикант, який закінчив у нього навчання;

– оцінити ментора за технологією може лише той практикант, котрий пройшов у нього навчання за цією технологією.

2.2 Архітектура програмного забезпечення

Дане програмне забезпечення складається з трьох основних частин: з клієнтської частини, серверної частини та бази даних. Розглянемо їх всі по черзі.

2.2.1 База даних.

Під час розробки даного програмного забезпечення була використана СКБД PostgreSQL 10. Детально опишемо структуру бази даних, використану в проекті.

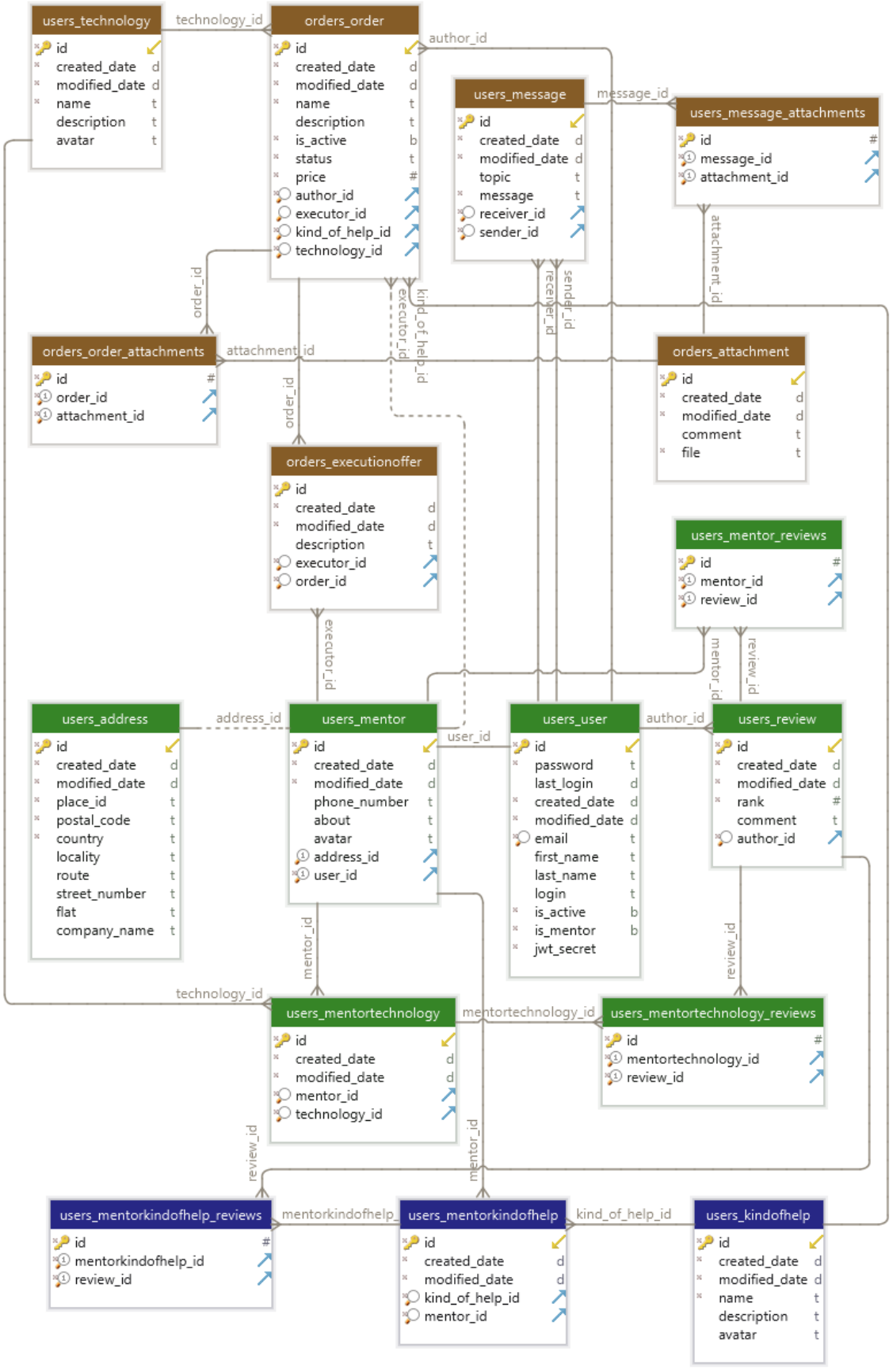


Рисунок 2.2 - Схема бази даних

Таблиця 2.1 - Короткий опис таблиць БД

Назва таблиці	Опис таблиці
orders_attachment	Список файлових вкладень, що прив'язані до замовлень або текстових повідомлень
orders_executionoffer	Заявки на виконання замовлень
orders_order	Замовлення
orders_order_attachments	Вкладення (файли) до замовлення
users_address	Адреса ментора
users_kindofhelp	Базові послуги, наявні у системі
users_mentor	Основні дані про ментора
users_mentor_reviews	Зв'язок між ментором та відгуками користувачів
users_mentorkindofhelp	Список послуг, якими володіє ментор
users_mentorkindofhelp_reviews	Зв'язок між послугою, якою володіє ментор та відгуками користувачів
users_mentortechnology	Список технологій, якими володіє ментор
users_mentortechnology_reviews	Зв'язок між технологією, якою володіє ментор та відгуками користувачів
users_message	Текстове повідомлення користувача
users_message_attachments	Вкладення (файли) до повідомлення користувача
users_review	Оцінка (відгук) користувача
users_technology	Базові технології, наявні у системі
users_user	Основні дані про користувача

Таблиця 2.2 - Службові поля БД (наявні у кожній таблиці)

Поле	Тип	Призначення
id	UUID	Унікальний ідентифікатор запису в таблиці
created_date	Timestamp	Дата створення запису в таблиці
modified_date	Timestamp	Дата останньої модифікації запису в таблиці

Таблиця 2.3 - Опис таблиці orders_attachment

Поле	Тип	Призначення
comment	Text	Текстовий коментар до прикріплення
file	Varchar(100)	Файл прикріплення

Таблиця 2.4 - Опис таблиці orders_executionoffer

Поле	Тип	Призначення
description	Text	Опис заявки на виконання замовлення
Executor_id	UUID	Посилання на виконавця
Order_id	UUID	Посилання на замовлення

Таблиця 2.5 - Опис таблиці orders_order

Поле	Тип	Призначення
name	Varchar(255)	Назва замовлення
description	Text	Опис замовлення
Is_active	Boolean	Прапорець, що дозволяє ховати замовлення від перегляду
status	Varchar(10)	Прапорець, що дозволяє визначити статус виконання замовлення
price	Integer	Ціна замовлення
Author_id	UUID	Посилання на автора
Executor_id	UUID	Посилання на виконавця
Kindofhelp_id	UUID	Посилання на вид тренінгу
Technology_id	UUID	Посилання на технологію

Таблиця 2.6 - Опис таблиці users_address

Поле	Тип	Призначення
Place_id	Varchar(200)	Посилання на координати GoogleMaps
Postal_code	Varchar(50)	Поштовий індекс
country	Varchar(200)	Країна
locality	Varchar(200)	Місто
route	Varchar(200)	Вулиця
Street_number	Varchar(50)	Номер будівлі
flat	Varchar(50)	Номер приміщення

Продовження таблиці 2.6

Company_name	Varchar(200)	Назва компанії

Таблиця 2.7 - Опис таблиці users_kindofhelp

Поле	**Тип**	**Призначення**
name	Varchar(255)	Назва виду тренінгу
description	Text	Опис виду тренінгу
avatar	Varchar(100)	Зображення

Таблиця 2.8 - Опис таблиці users_mentor

Поле	**Тип**	**Призначення**
Phone_number	Varchar(128)	Номер телефону
about	Text	«Про себе»
avatar	Varchar(100)	Зображення
Address_id	UUID	Посилання на адресу
User_id	UUID	Посилання на користувача

Таблиця 2.9 - Опис таблиці users_message

Поле	**Тип**	**Призначення**
topic	Varchar(255)	Тема повідомлення
message	Text	Текст повідомлення
Receiver_id	UUID	Посилання на отримувача
Sender_id	UUID	Посилання на відправника

Таблиця 2.10 - Опис таблиці users_review

Поле	Тип	Призначення
rank	Integer	Оцінка від 1 до 10
comment	Text	Текстовий коментар
Author_id	UUID	Посилання на автора

Таблиця 2.11 - Опис таблиці users_technology

Поле	Тип	Призначення
name	Varchar(255)	Назва технології
description	Text	Опис технології
avatar	Varchar(100)	Зображення

Таблиця 2.12 - Опис таблиці users_user

Поле	Тип	Призначення
email	Character varying(255)	Поштова адреса
First_name	Character varying(30)	Ім’я
Last_name	Character varying(30)	Прізвище
login	Character varying(30)	Альтернативне ім’я
Is_active	Boolean	Прапорець, що дозволяє відключати користувачів від системи
Is_mentor	Boolean	Прапорець, що визначає, чи має користувач статус ментора
JWT_secret	UUID	Ключ для сесійної авторизації

2.2.2 Серверна частина.

Архітектура Django, як і у більшості веб-фреймворків, дуже схожа на MVC, проте в якості контроллера виступає сам фреймворк: саме він, без участі розробника, зв’язує моделі з їх представленнями.

Розглянемо структуру проекту серверної частини веб-застосунку.

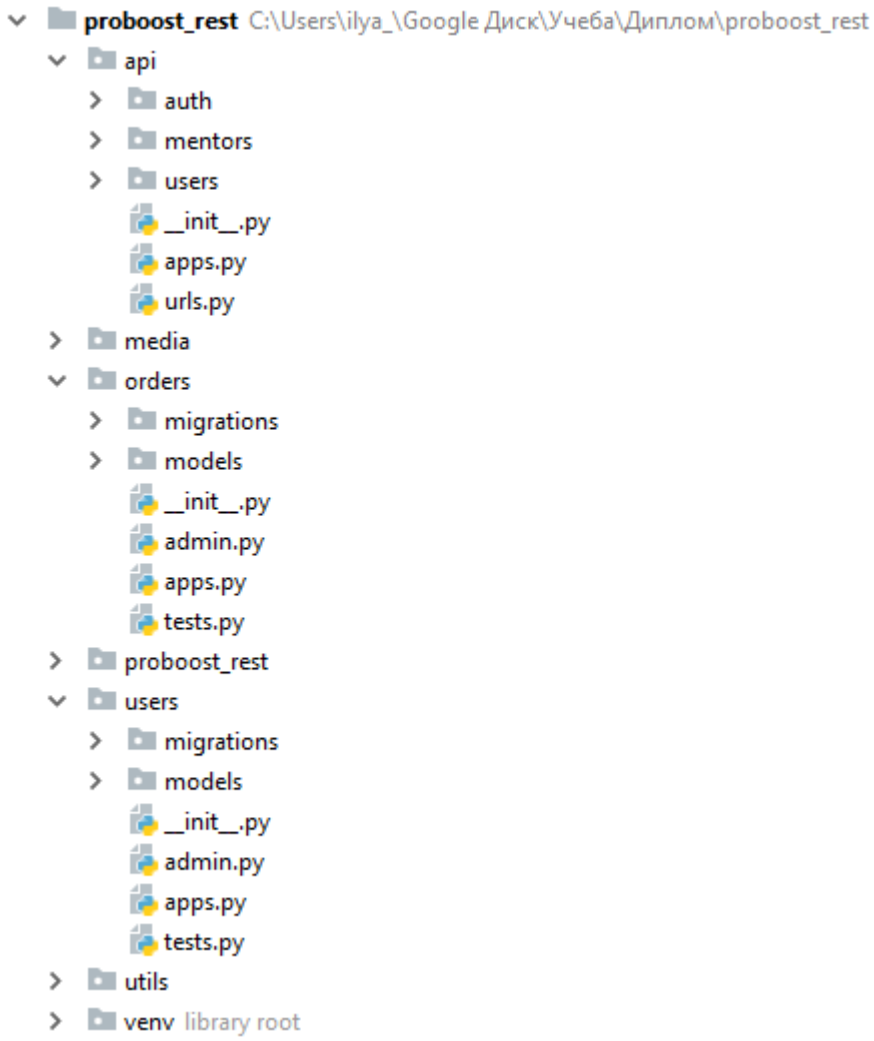


Рисунок 2.3 - Структура проекту серверної частини

Проект складається з модулів, кожен з яких ізольований та може без проблем застосовуватися в інших проектах.

Розглянемо ці модулі більш детально.

Таблиця 2.13 - Призначення модулів

Назва модулю	Призначення
api	Тут знаходяться всі представлення моделей (Views у MVC) та всі URL, що ведуть до цих представлень. Також саме тут здійснюється валідація та перетворення даних.
media	Тут зберігаються всі медіа-файли, посилання на які знаходяться в базі даних. Призначений лише для розробки та тестування. Для звичайної роботи веб-застосунку повинен використовуватися сторонній сервіс (наприклад, Amazon S3).
orders	Тут знаходяться всі моделі (Models у MVC), що мають відношення до замовлень на навчання.
proboost_rest	Кореневий модуль проекту. Саме тут здійснюються підключення і налаштування інших модулів.
users	Тут знаходяться всі моделі, що мають відношення до даних про користувачів.
utils	Тут знаходяться допоміжні класи та функції, що призначені для повторного використання та запобігають дублюванню програмного коду.
venv	Тут зберігаються всі сторонні модулі мови Python, які використовуються даним веб-застосунком. Вони ізольовані від бібліотек, що встановлені глобально на комп’ютері.

Модулі orders та users мають схожу структуру. Вони складаються з, власне, набору моделей, які є інтерфейсами з редагування відповідних таблиць в базі даних, набору міграцій до бази даних, та тестів.

Моделі являють собою класи, що мають в якості властивостей поля, ідентичні полям в базі даних. На етапі створення моделей визначаються назви таблиць в базі даних, їх поля, типи та залежності між даними в різних таблицях.

```
class Mentor(BaseModel):
    """
    The Mentor object:
    An addition to user model for use by mentors.
    """
    user = models.OneToOneField(verbose_name='User',
                                to='users.User',
                                on_delete=models.CASCADE,
                                related_name='mentor')
    phone_number = models.CharField(verbose_name='Phone Number',
                                    max_length=128,
                                    blank=True,
                                    null=True,
                                    help_text='Input phone number in international format here',
                                    validators=[RegexValidator(regex='^\+[1-9]\d{4,14}$',
                                                               message='Input phone number in '
                                                                       'international format `E.164`')])
    address = models.OneToOneField(verbose_name='Address',
                                   to='users.Address',
                                   related_name='user',
                                   null=True,
                                   blank=True,
                                   on_delete=models.CASCADE)
    about = models.TextField(verbose_name='About me',
                             max_length=300,
                             null=True,
                             blank=True)
    avatar = models.ImageField(upload_to=get_image_path, blank=True, null=True)
    reviews = models.ManyToManyField(verbose_name='Reviews',
                                     to='users.Review',
                                     blank=True,
                                     null=True,
                                     related_name='mentor_reviews')
```

Рисунок 2.4 - Приклад класу-моделі

Всі класи-моделі наслідуються від базового класу-моделі, що прописаний у бібліотеці Django. Кожне поле такого класу є спеціальним об’єктом – полем, що і визначає тип цього поля в базі даних, параметри валідації та інші властивості.

Міграції – це також класи, які призначені для відслідковування та підтримки відповідності структури бази даних та відповідних моделей. Тобто,

якщо програмний код оновився разом із моделями, то зовсім не обов'язково створювати базу даних з нуля чи вручну редагувати її структуру. Достатньо лише написати міграцію та виконати його: фреймворк зробить це автоматично за наявності підключеної бази даних.

```
class Migration(migrations.Migration):

    dependencies = [
        ('users', '0004_auto_20190526_1129'),
    ]

    operations = [
        migrations.AlterField(
            model_name='mentor',
            name='reviews',
            field=models.ManyToManyField(blank=True, null=True,
                                         related_name='mentor_reviews', to='users.Review', verbose_name='Reviews'),
        ),
    ]
```

Рисунок 2.5 - Приклад класу-міграції

Сама міграція містить у собі назву попередньої міграції, від якої вона залежить (до її виконання ця міграція не може бути виконана) та набір операцій, що мають бути виконані над базою даних, щоб зробити її актуальною для даної міграції. Зазвичай ці класи створюються автоматично, проте це може бути зроблено і самим програмістом.

Модуль api є ключовим для серверної частини веб-застосунку. Саме він перенаправляє запити до необхідних представлень та займається їх обробкою. Структура модулю є ієрархічною: кожен з модулів розбитий на менші модулі, які займаються обробкою вужчого спекрту запитів.

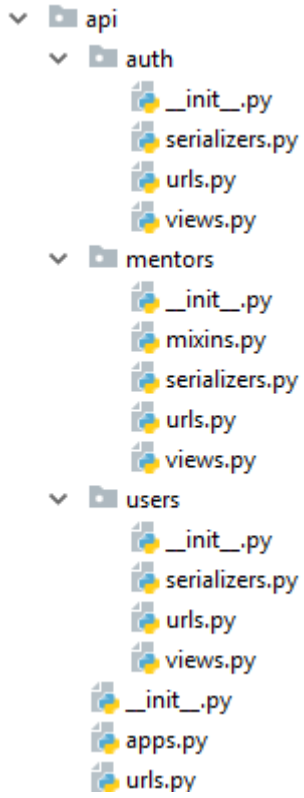


Рисунок 2.6 - Структура модулю api

Таблиця 2.14 - Призначення файлів у модулі api

Назва файлу	**Призначення**
__init__.py	Службовий файл. Використовується для підключення даного модулю до інших.
apps.py	Службовий файл. Використовується для ініціалізації модулю.
urls.py	Тут зберігаються списки URL-шаблонів, та відповідні їм представлення, на які перенаправляються запити, що відповідають шаблону.

Продовження таблиці 2.14

views.py	Тут зберігаються списки класів-представлень, що включають в себе валідацію вхідних запитів, перевірку прав доступу до даних у представленні та модифікацію даних (за потребою). Також саме класи-представлення визначають набір даних, з якими буде працювати вхідний HTTP-запит на сервер.
mixins.py	Тут зберігаються так звані класи-домішки. Їх призначення – це організація коду, що дублюється в представленнях, в окремі класи, від яких потім наслідуються класи-представлення. Це можливо завдяки підтримки мовою програмування Python множинного наслідування.
serializers.py	Тут зберігаються так звані класи-серіалізаори. На відміну від класів-представлень, вони використовуються для визначення набору полів, які доступні для читання та запису для даного запиту. Також вони дозволяють обробляти вкладені набори даних, та зберігати ці дані у різних моделях, автоматично створюючи всі необхідні зв’язки між ними.

2.2.3 Клієнтська частина.

Для розробки клієнтського додатку, згідно ТЗ, був використаний фреймворк Angular 7. В основі даного фреймворку також використовується архітектура MVC. В даному випадку роль представлення грають HTML-

шаблони, роль контроллерів – веб-компоненти, а роль моделі – різноманітні сервіси, що здійснюють запити до бази даних.

Розглянемо детальніше структуру проекту.

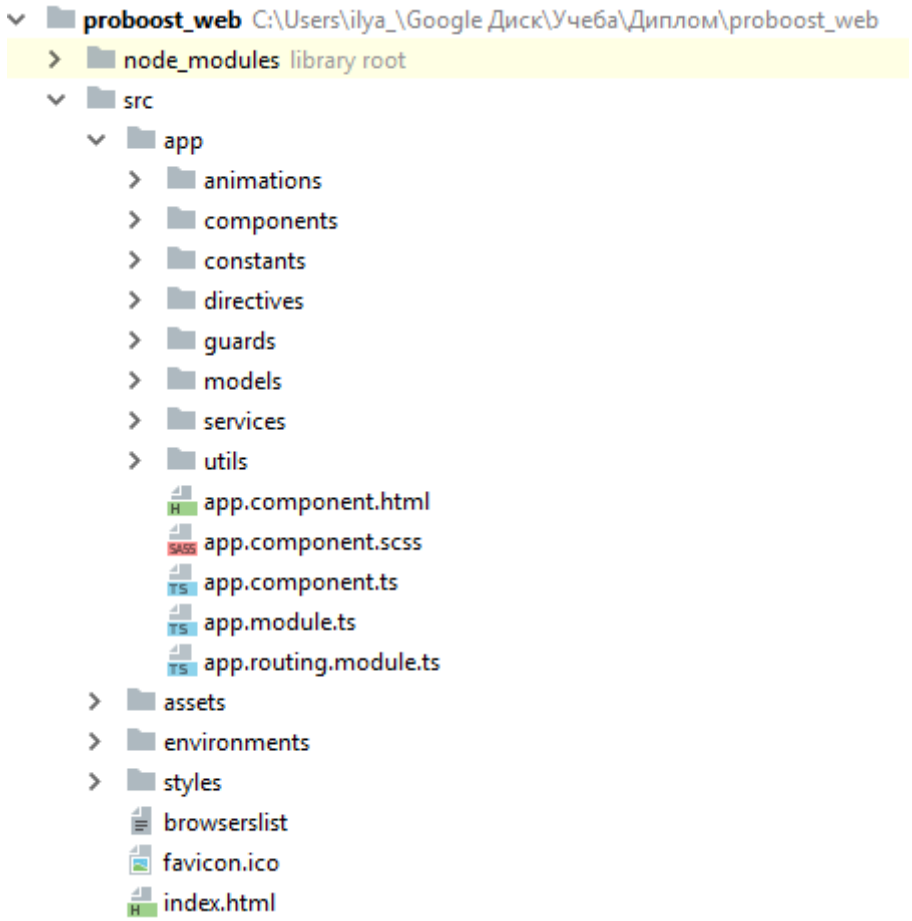


Рисунок 2.7 - Структура проекту клієнтської частини

Проект складається з набору директорій, що містять код різноманітного призначення. Вони не є ізольованими програмними модулями та залежать один від одного, тож не можуть бути просто переміщені з одного проекту в інший.

Таблиця 2.15 - Призначення директорій

Назва директорії	Призначення
node_modules	Папка для зберігання npm – модулів, які використовуються даним веб-застосуванням
src	Коренева папка проекту

Продовження таблиці 2.15

App	Коренева директорія для всіх складових Angular-додатку
assets	Папка, у якій містяться всі медіа-файли, що використовуються даним додатком (зображення, шрифти, стилі тощо)
environments	Містить файли зі змінними оточення (конфігурація для запуску у режимі розробки, у тестовому режимі та у режимі готового продукту)
styles	Папка, що містить набір файлів із глобальними стилями для програмного забезпечення
index.html	«Головний» файл, що слугує для запуску Angular-додатку

Як ми бачимо зі структури клієнтської частини, складових Angular-додатку, що складають його основу, теж є декілька видів. Розглянемо їх детальніше.

Таблиця 2.16 – Складові Angular-додатку

Назва	**Призначення**
animations	Анімації. Представляють собою файли конфігурації для застосування у Angular-компонентах. Фреймворк використовує власний анімаційний движок, тому цього достатньо для коректної роботи анімацій.

Продовження таблиці 2.16

Components	Компоненти. Є «будівельними блоками» для Angular-застосунку. Кожен такий блок складається з файлу-представлення, що є HTML – шаблоном для відображення даних, файлу зі стилями для цього шаблону, та класу компоненту, що є контроллером в даній MVC-моделі. Головною особливістю Angular-компонентів є можливість вкладати їх один в інший. Таким чином, все веб-застосування складається з невеликих компонент, що об'єднані у великі компоненти. Кореневиим компонентом в даному проекті є наступні файли: app.component.ts (~.scss, ~.html). Зазвичай компоненти групуються в модулі. Кореневим модулем є app.module.ts
constants	Константи. Директорія для збереження основних налаштувань веб-застосування.
directives	Директиви. Є дуже схожими до Angular-компонент за своєю структурою, проте не мають власного представлення, а слугують лише для розширення функціоналу існуючих компонент. Вбудовуються в них у HTML-шаблонах.
guards	«Охоронці». Класи, що слугують для контролю доступу в різних частинах веб-застосування. Є складовою частиною модулів веб-компонент.

Продовження таблиці 2.16

Models	Моделі. Є класами, що організують неформатовані дані, що надходять з серверу у Typescript-об’єкти. Використовуються здебільшого для контролю типів на стороні клієнту. Зазвичай є просто копіями моделей серверної частини веб-застосування.
services	Сервіси. Класи, що реалізують інтерфейси по обробці даних у всьому застосунку. Зазвичай являють собою набори асинхронних методів для здійснення запитів на сервер та повернення даних компонентам, що їх викликають. Вбудовуються у контроллери компонент за допомогою dependency injection.
utils	Подібно серверній частині, тут знаходяться допоміжні класи та функції, що призначені для повторного використання та запобігають дублюванню програмного коду.

Окрім розділення на різні за призначенням складові веб-застосування, його компоненти розділені ще й на модулі.

Рисунок 2.8 - Модулі Angular-застосування

Кожне веб-застосування, створене за допомогою даного фреймворку, містить щонайменше один кореневий модуль (згаданий вище app.module.ts). Ці модулі об’єднують в собі компоненти та сервіси, що мають бути завантажені одночасно. Тож у великих веб-застосунках для підвищення швидкості роботи слід розбивати великі модулі на менші. Окрім пришвидшення роботи застосунку, це ще й покращить наочність структури проекту.

Кожен такий модуль містить:

– класи компонент, що можуть використовуватися даним модулем;

– класи сервісів, що можуть використовуватися даним модулем;

– клас-маршрутизатор.

```
export const routes: Routes = [
    {
        path: '', component: LoginComponent, data: { title: getTitle },
        children: [
            { path: '', component: LoginFormComponent, pathMatch: 'full' },
            { path: 'register', component: RegisterFormComponent },
            { path: 'forgot', component: ForgotFormComponent }
        ]
    }
];

@NgModule({
    imports: [RouterModule.forChild(routes)],
    exports: [RouterModule]
})
export class LoginRoutingModule {
}
```

Рисунок 2.9 - Приклад класу-маршрутизатору

Клас-маршрутизатор слугує для задання правил відображення різних компонент в залежності від URL-маршруту, за яким звертаються до веб-застосування.

Оскільки Angular слугує для побудови SPA (Single Page Application), тобто, включає в себе лише один веб-застосунок, який буде запущений користувачем при переході за будь-яким маршрутом, то йому необхідні засоби для імітації роботи MPA (Multi Page Application). Класи-маршрутизатори виконують саме таку задачу.

Кореневий компонент модуля може включати в себе спеціальний компонент router-outlet, на місце якого буде вставлений той компонент, що відповідає поточному маршруту. Така навігація може мати як завгодно багато рівнів вкладеності.

2.3 Конструювання програмного забезпечення

2.3.1 Серверна частина

Далі буде наведений короткий опис основних класів серверної частини веб-застосування.

Таблиця 2.14 - Опис класів серверної частини

Клас	Батьківський клас	Опис
BaseModel	django.db.models.Model	Базовий клас-представлення для даних в основних таблицях БД. Включає в себе універсальні поля id (ідентифікатор запису в бд), created_date (дата створення запису в бд), modified_date(дата останньої модифікації запису в бд)
Attachment	BaseModel	Клас-модель для даних в таблиці orders_attachment
Executionoffer	BaseModel	Клас-модель для даних в таблиці orders_executionoffer
Order	BaseModel	Клас-модель для даних в таблиці orders_order
Address	BaseModel	Клас-модель для даних в таблиці users_address
Kindofhelp	BaseModel	Клас-модель для даних в таблиці users_kindofhelp
Mentor	BaseModel	Клас-модель для даних в таблиці users_mentor
Message	BaseModel	Клас-модель для даних в таблиці users_message

Продовження таблиці 2.17

Review	BaseModel	Клас-модель для даних в таблиці users_message_attachments
Technology	BaseModel	Клас-модель для даних в таблиці users_review
User	BaseModel	Клас-модель для даних в таблиці users_technology
AttachmentSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Attachment
ExecutionofferSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Executionoffer
OrderSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Order
AddressSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Address
KindofhelpSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Kindofhelp
MentorSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Mentor

Продовження таблиці 2.17

MessageSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Message
ReviewSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Review
TechnologySerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі Technology
UserSerializer	serializers.ModelSerializer	Клас-серіалізатор для відображення даних моделі User
UserLogoutAllView	rest_framework. views.APIView	Клас-представлення для запиту на вихід користувача з системи
SearcherAndOrderQuerySetMixin	-	Клас-домішка для фільтрації даних у класах-представленнях
MentorsFilterOrderMixin	SearcherAndOrderQuerySetMixin	Клас-домішка для фільтрації даних у класах-представленнях, що працюють з моделлю менторів
MentorsQuerySetMixin	-	Клас-домішка для вибірки даних, доступних для обробки класом-представленням менторів

Продовження таблиці 2.17

AuthMixin	-	Клас-домішка для перевірки авторизації користувача, що отримує доступ до класа-представлення
UserAuthMixin	-	Клас-домішка для перевірки авторизації користувача, що намагається редагувати власні дані
MentorAuthMixin	-	Клас-домішка для перевірки авторизації користувача - ментора, що отримує доступ до класа-представлення
MentorPermission	rest_framework.permissions.BasePermission	Клас-валідатор, що перевіряє, чи має поточний користувач статус ментора
UserRoutesPermission	rest_framework.permissions.BasePermission	Клас-валідатор, що перевіряє, чи має поточний користувач право на редагування іншого користувача (дійсне лише для самого себе)

Продовження таблиці 2.17

UserPermission	rest_framework.permissions.BasePermission	Клас-валідатор, що перевіряє, чи авторизований поточний користувач
MentorsView	MentorsFilterOrderMixin, MentorsQuerySetMixin, SerializerPerActionAPIViewMixin, AuthMixin, ListAPIView	Клас-представлення усього переліку менторів
MentorDetailView	MentorsFilterOrderMixin, MentorsQuerySetMixin, SerializerPerActionAPIViewMixin, AuthMixin, RetrieveUpdateAPIView	Клас-представлення конкретного ментора
SelfMentorView	MentorAuthMixin, RetrieveUpdateAPIView	Клас-представлення ментора поточного користувача
SelfMentorCreateView	AuthMixin, CreateAPIView	Клас-представлення для створення ментора поточного користувача

Продовження таблиці 2.17

OrdersView	OrdersFilterOrderMixin, OrdersQuerySetMixin, SerializerPerActionAPIViewMixin, AuthMixin, ListAPIView	Клас-представлення усього переліку замовлень
OrderDetailView	OrdersFilterOrderMixin, OrdersQuerySetMixin, SerializerPerActionAPIViewMixin, AuthMixin, RetrieveUpdateAPIView	Клас-представлення конкретного замовлення
OrdersFilterOrderMixin	SearcherAndOrderQuerySetMixin	Клас-домішка для фільтрації даних у класах-представленнях, що працюють з моделлю замовлень
OrdersQuerySetMixin	-	Клас-домішка для вибірки даних, доступних для обробки класом-представленням замовлень

Продовження таблиці 2.17

ExecutionOffersView	ExecutionOffersFilterOrderMixin, ExecutionOffersQuerySetMixin, SerializerPerActionAPIViewMixin, AuthMixin, ListAPIView	Клас-представлення усього переліку заявок на виконання замовлень
ExecutionOfferDetailView	ExecutionOffersFilterOrderMixin, ExecutionOffersQuerySetMixin, SerializerPerActionAPIViewMixin, AuthMixin, RetrieveUpdateAPIView	Клас-представлення конкретної заявки на виконання замовлення
ExecutionOffersFilterOrderMixin	SearcherAndOrderQuerySetMixin	Клас-домішка для фільтрації даних у класах-представленнях, що працюють з моделлю заявок на виконання замовлень
ExecutionOffersQuerySetMixin	-	Клас-домішка для вибірки даних, доступних для обробки класом-представленням заявок на виконання замовлень

2.3.2 Клієнтська частина

Далі буде наведений короткий опис основних модулів клієнтської частина веб-застосування та їх складових.

Таблиця 2.18 - Опис модулів клієнтської частини

Модуль	**Кореневий модуль**	**Опис**
AppModule	-	Кореневий модуль проекту
LoginModule	AppModule	Модуль що відповідає за авторизацію та реєстрацію користувача в системі
ResetPasswordModule	AppModule	Модуль що відповідає за відновлення паролю користувача в системі
MainModule	AppModule	Кореневий модуль для основної частини веб-застосування
AccountModule	MainModule	Модуль редагування персональних даних користувача
AccountMentorModule	MainModule	Модуль редагування персональних даних ментора
SharedModule	-	Модуль, що містить компоненти, які використовуються всіма вище переліченими модулями
ToastModule	-	Модуль що відповідає за відображення інформаційних повідомлень

Продовження таблиці 2.18

FormsModule	-	Модуль, що відповідає за обробку даних форм вводу-виводу

Таблиця 2.19 - Опис сервісів клієнтської частини

Сервіс	**Опис**
BaseHttpService	Сервіс що містить базові методи для створення і обробки HTTP-запитів
Interceptor	Сервіс, що «перехоплює» всі HTTP-запити, що завершилися з помилкою аутентифікації та перенаправляє користувача на сторінку авторизації в такому випадку.
LoginService	Сервіс, що містить методи для роботи з даними користувача
MentorService	Сервіс, що містить методи для роботи з даними менторів
OrderService	Сервіс, що містить методи для роботи з даними замовлень
ExecutionOfferingService	Сервіс, що містить методи для роботи з даними заявок на виконання замовлень
ModalService	Сервіс, що займається відображенням на екрані спливаючих вікон
PopoverService	Сервіс, що займається відображенням на екрані спливаючих меню
RegisterIconsService	Сервіс, що займається реєстрацією спрощених назв для зображень

Таблиця 2.20 - Опис компонент клієнтської частини

Компонент	Модуль	Опис
AppComponent	AppModule	Кореневий компонент веб-застосунку
MainComponent	MainModule	Кореневий компонент головної частини веб-застосунку
SidebarComponent	MainModule	Бічна панель керування
FooterComponent	MainModule	Нижня інформаційна панель
MentorsComponent	MainModule	Сторінка для відображення менторів
ListMentorsComponent	MainModule	Список менторів
MentorDetailsComponent	MainModule	Детальна інформація про менторів
OrdersComponent	MainModule	Сторінка для відображення замовлень
MyOrdersComponent	MainModule	Сторінка для відображення заявок на виконання замовлень
LoginComponent	LoginModule	Сторінка входу/реєстрації в системі
LoginFormComponent	LoginModule	Форма входу
RegisterFormComponent	LoginModule	Форма реєстрації
ForgotFormComponent	LoginModule	Форма відновлення паролю
PortalInputComponent	LoginModule	Текстове поле для вводу
ResetPasswordComponent	ResetPasswordModule	Сторінка відновлення паролю

Продовження таблиці 2.20

AccountComponent	AccountModule	Сторінка профілю користувача
AccountSettingsComponent	AccountModule	Дані профілю користувача
AccountMentorComponent	AccountMentorModule	Сторінка профілю ментора
AccountMentorSettingsComponent	AccountMentorModule	Дані профілю ментора
CoreButtonComponent	SharedComponentsModule	Кнопка
CoreLinkComponent	SharedComponentsModule	Посилання
HeaderComponent	SharedComponentsModule	Заголовок із полем для пошуку
InputComponent	SharedComponentsModule	Текстове поле для вводу
FileUploadComponent	SharedComponentsModule	Поле для завантаження файлів
TableComponent	SharedComponentsModule	Універсальний конструктор таблиць
SwitchComponent	SharedComponentsModule	Перемикач
SwitchLinkedComponent	SharedComponentsModule	Перемикач
CheckboxComponent	SharedComponentsModule	Перемикач

Продовження таблиці 2.20

SpinnerComponent	SharedComponentsModule	Елемент анімації завантаження
MoreMenuComponent	SharedComponentsModule	Елемент розгортуючогося списку
SimpleHeaderComponent	SharedComponentsModule	Заголовок
SearchComponent	SharedComponentsModule	Елемент для пошуку (поле для вводу)

2.4 Аналіз безпеки даних

Веб-фреймворк Django за замовчуванням обладнаний більшістю необхідних механізмів захисту даних. Оглянемо їх нижче.

Для збереження паролей Django використовує алгоритм PBKDF2 з хешем SHA256. Цей алгоритм збереження паролей є рекомендованим Національним інститутом стандартів і технології (NIST). Алгоритм передбачає перетворення будь-якого вхідного паролю користувача на псевдовипадковий ключ, час зворотнього підбору якого є достатньо великим і можливий лише за допомогою повного перебору всіх можливих вхідних паролей.

Аутентифікація користувача виконується за допомогою сессійного JWT – токена – ключа, який сервер видає авторизованому користувачеві на час сессії (10 хвилин), та який клієнт має передавати з кожним HTTP – запитом на ресурс, з обмеженим доступом. Інакше сервер видасть помилку HTTP 401 Unauthorized.

Також Django за замовчуванням включає в себе захист від найбільш розповсюджених типів атак, а саме:

– захист від SQL-ін’єкцій. Здійснює валідацію параметрів запиту, що передаються користувачем та не дозволяє виконувати SQL-запити на сервері, що передаються в цих параметрах;

– SSL/HTTPS. Даний протокол використовується для шифрування всього вхідного та вихідного трафіку, що захищає дані від Man-in-the-middle атак;

– валідація заголовку HOST. Використовується для перевірки джерела HTTP-запитів на сервер. Приймаються запити лише від довірених хостів.

2.5 Висновки по розділу

В даному розділі був описаний процес моделювання програмного забезпечення, його архітектура та деякі особливості реалізації. Також був проведений детальний аналіз безпеки даних веб-застосування.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Протягом життєвого циклу даного програмного продукту будуть виникати дефекти та помилки. Вони можуть проявлятися на різних етапах розробки: під час проектування програмного забезпечення, під час його розробки, під час його активного використання у період підтримки.

Причинами їх виникнення можуть бути:

- помилки програміста;
- незадовільнені вимоги експлуатації ПЗ.

В будь-якому випадку наслідками їх виникнення будуть втрачені гроші замовника та втрачені користувачі веб-застосування. Щоб запобігти таким випадкам, потрібна сувора система контролю якості програмного забезпечення. В даному проекті був використаний підхід тестування ПЗ.

Найбільш ефективним способом розробки стратегії тестування є написання тест-плану до програмного продукту за стандартом IEEE 829, що включає в себе як стратегію тестування продукту, так і ризики, що з нею пов’язані.

3.2 Опис процесів тестування

Далі буде наведений тест-план із детальним описом процесів тестування.

3.2.1 Ідентифікатор тест-плану

PRO_boost_test_plan.

3.2.2 Вступ до тест-плану

Це веб-застосунок, написаний на Django з використанням Django REST Framework (серверна частина) та TypeScript з використанням Angular 7+ Framework (клієнтська частина). Проект включає в себе наступні модулі:

- модуль реєстрації;
- модуль входу/виходу з системи та відновлення паролю;
- модуль облікового запису клієнта;
- модуль облікового запису майстра;
- головний модуль (створення, пошук замовлень, пошук та відображення майстрів);
- модуль обміну текстовими повідомленнями (чат);
- система відгуків та оцінювання майстрів.

3.2.3 Об’єкт тестування

- Модуль реєстрації: версія 0.1.0.
- Модуль входу/виходу з системи та відновлення паролю: версія 0.1.0.
- Модуль облікового запису клієнта: версія 0.2.0.
- Модуль облікового запису майстра: версія 0.2.0.
- Головний модуль (створення, пошук замовлень, пошук та відображення майстрів) : версія 0.1.0.
- Модуль обміну текстовими повідомленнями (чат) : версія 0.3.0.
- Система відгуків та оцінювання майстрів: версія 1.1.0.

3.2.4 Компоненти, що тестуються

- Реєстрація як клієнта.
- Реєстрація як майстра.
- Вхід до системи.

- Вихід з системи.
- Відновлення паролю.
- Пошук майстра.
- Пошук з використанням фільтрації.
- Спілкування клієнта й майстра з використанням текстового чату.
- Редагування облікового запису клієнта.
- Редагування облікового запису майстра.

3.2.5 Компоненти, що не тестуються

- Система відгуків та оцінювання майстрів: не входить до MVP (minimum viable product).

3.2.6 Підхід

Тестування для проекту PROboost складатиметься з юніт-, інтеграційного, системного (функціональне, GUI (Graphic User Interface), навантажувальне) тестування та UAT (User Acceptance Testing):

а) юніт-тестування. Буде перевіряти окремі компоненти разом з їх функціями в ізоляції методом чорної скриньки. Буде задіяне лише на серверній частині проекту. Буде автоматизоване за допомогою засобів Django та буде налаштований процес автоматичного запуску тестів під час кожної збірки програми за допомогою Jenkins CI. Автоматичне тестування виконуватиметься через низьку складність тестів, велику їх кількість та велику частоту запусків. (Будуть запускатися регресивно для перевірки на працездатність вже завершених модулів під час розробки) Виконуватиметься розробником;

б) інтеграційне тестування. Буде перевіряти взаємодію модулів між собою методом чорної скриньки, перевіряючи коректність роботи програмного інтерфейсу серверної частини. Буде вводитися по мірі реалізації модулів, після виявлення та виправлення всіх критичних дефектів на попередньому етапі. Також буде проведене наповнення тестової бази даних для цього тестування.

Буде автоматизоване за допомогою Mocha та буде налаштований процес автоматичного запуску тестів під час кожної збірки програми за допомогою Jenkins CI. Автоматичне тестування виконуватиметься через високу складність тестів для виконання людиною (Через необхідність створювати заново тестову базу даних після кожної серії тестів). Виконуватиметься розробником;

в) системне тестування:

1) функціональне тестування. Буде перевіряти відповідність функціоналу програмного забезпечення заданим специфікаціям методом чорної скриньки. Виконуватиметься тестувальником після успішного виконання попереднього етапу. В разі невдачі програма буде повернена на доопрацювання. Проводитиметься в ручному режимі, через високу складність тестів та невелику їх кількість;

2) GUI (Graphic User Interface) тестування. Буде перевіряти відповідність інтерфейсу програмного забезпечення заданим вимогам. Виконуватиметься тестувальником паралельно із попереднім етапом. В разі невдачі програма буде повернена на доопрацювання. Проводитиметься в ручному режимі, через високу складність тестів для автоматизації;

3) навантажувальне тестування. Буде перевіряти відповідність граничних меж навантаження на програмне забезпечення заданим вимогам. Виконуватиметься тестувальником після всіх попередніх етапом. В разі невдачі програма буде повернена на доопрацювання та оптимізацію. Проводитиметься в автоматизованому режимі, оскільки одна людина фізично не в змозі перевірити можливі межі навантаження на сервер;

г) UAT (User Acceptance Testing). Перевірка виконання всіх бізнес-вимог до програмного продукту. Фінальна перевірка взаємодії всіх програмних модулів. Виконуватиметься замовником. В разі невдачі повернутися до

системного тестування. В разі успіху – визнати проект повністю завершеним та перейти до його релізу. Проводитиметься в ручному режимі.

3.2.7 Критерії проходження тестів

а) Повне проходження: Система буде ухвалена без змін. Система підтримує даний бізнес-сценарій. Ніяких неочікуваних результатів виявлено не було.

б) Часткове проходження: Система підтримує даний бізнес-сценарій, але у максимум 10% випадках трапляються помилки, що ускладнюють перевірку тестового випадку або призводять до результату, відмінного від очікуваного.

в) Відхилення: Система не підтримує даний бізнес-сценарій або неочікувано завершується/зависає під час тестування.

3.2.8 Критерії призупинення та продовження тестування

а) Недоступність AWS сервера протягом тестування.

Критерій продовження: AWS сервер, на якому запущений програмний продукт, стає доступним.

б) Критична помилка не дозволяє подальшого тестування.

Критерій продовження: Критична помилка усунена та програма з останніми змінами запущена на AWS сервері.

в) Не було дотримано строків розробки функціоналу, що тестується.

Критерій продовження: Весь функціонал, що має вищий пріоритет, завершений і протестований.

3.2.9 Результати проведення тестування

а) Тест план.

б) Тест кейси для юніт-, інтеграційного та системного тестування (функціональні).

в) Звіти про помилки.

г) Підсумкові звіти тестування.

3.2.10 Задачі для проведення тестування

Таблиця 3.1 - Задачі для проведення тестування

Задача	**Відповідальний**
Написання тест-плану	Менеджер проекту
Написання тест кейів для юніт-, інтеграційного та системного тестування	Тестувальник
Зробити звіти про помилки	Тестувальник
Зробити підсумкові звіти	Тестувальник
Написати юніт- тести, інтегрувати їх виконання у процес розробки за допомогою Jenkins CI	Розробник
Написати інтеграційні тести, інтегрувати їх виконання у процес розробки за допомогою Jenkins CI	Розробник
Виконати юніт-, інтеграційне, системне тестування (функціональне, користувацького інтерфейсу, навантажувальне)	Тестувальник
Виконати UAT	Замовник

3.2.11 Технічні потреби

Таблиця 3.2 - Технічні потреби

Потреби	**Мінімальні**	**Рекомендовані**
Апаратні	PC with 8GB RAM i3 5300	PC with 16GB RAM i5 7500
Програмні	Webstorm, Pycharm, Python 3.6.6, Docker, PostgreSQL, Selenium/Nightwatch, Chrome	Webstorm, Pycharm, Python 3.6.6, Docker, PostgreSQL, Selenium/Nightwatch, Chrome, IE, Edge, Firefox, Opera, Safari

3.2.12 Обов’язки

Таблиця 3.3 - Обов’язки та відповідальні особи

Обов’язок	**Відповідальний**
Управління процесами в проекті	Менеджер проекту
Написання юніт- та інтеграційних тестів	Розробник
Підготовка та виконання тестів	Тестувальник
Оцінка проекту	Менеджер проекту
Перевірка якості проекту	Тестувальник
Виправлення помилок	Розробник
Надання всіх технічних потреб	Менеджер проекту
Прийом виконаних робіт	Замовник

3.2.13 Необхідні компетенції та тренінги

Таблиця 3.4 - Необхідні навички

Навичка	**Член команди**
Програмування на Python	Розробник
Програмування на JavaScript	Розробник
Знання Django	Розробник
Знання Angular 2+	Розробник
Знання Selenium/Nightwatch	Тестувальник
Знання PostgreSQL	Розробник
Вміння публікувати веб-застосунки за допомогою AWS	Менеджер проекту
Знання Jenkins CI	Менеджер проекту
Управління проектом	Менеджер проекту
Написання модульних тестів	Тестувальник

Таблиця 3.5 - Необхідні тренінги

Тренінг	**Член команди**
AWS deployment course	Менеджер проекту
Jenkins CI deployment course	Менеджер проекту
Django REST Framework course	Розробник

3.2.14 Розклад

Таблиця 3.6 - Етапи тестування

Етап	**Час**
Написання тест-плану	01.12.2018 - 22.12.2018 (16 людино-годин)
Написання тест кейів для юніт-, інтеграційного та системного тестування	10.01.2019 - 20.01.2019 (24 людино-години)
Звіти про помилки	20. 01.2019 - 25. 01.2019 (8 людино-годин)
Підсумкові звіти	25.01.2019 - 01.02.2019 (8 людино-годин)
Юніт-тестування	01.02.2019 - 10.03.2019 (48 людино-годин)
Інтеграційне тестування	10.03.2019 - 21.03.2019 (24 людино-години)
Системне тестування	21.03.2019 - 14.04.2019 (24 людино-години)
UAT	14.04.2019 - 30.04.2019 (24 людино-години)

3.2.15 Ризики

Таблиця 3.7 - Ризики

Ризик	**Засіб протидії**	**Відповідальний**
Затримка розробки модулів для тестування	Залучити додаткові ресурси, перерозподілити пріоритети задач	Менеджер проекту
Пошкодження даних у базі даних, що унеможливлює подальшу розробку	Відтворити останню збережену версію бази	Розробник
Додавання нових вимог	Повідомити, що на етапі MVP додавання нових вимог неможливе. Узгодити подальшу співпрацю із замовником за додаткові кошти.	Менеджер проекту

3.3 Опис контрольного прикладу

В якості контрольного прикладу нижче буде наведений один з можливих тест-кейсів для даного програмного застосунку.

Таблиця 3.8 - Приклад тест-кейсу

Ідентифікатор тест-кейсу	Login_test_wrong_password
Назва тест-кейсу	Вхід до системи за допомогою неправильного паролю
Вхідні дані, необхідні для тестування	Логін користувача, рареєстрованого в системі
Передумови	Користувач зайшов на сторінку входу / реєстрації користувачів
Кроки виконання	1. Ввести в поле для логіну користувача логін користувача, рареєстрованого в системі. 2. Ввести в поле для паролю користувача будь-яку послідовність символів. 3. Натиснути кнопку «Sign In»
Очікуваний результат	Користувач не зможе увійти в систему. Йому буде показане повідомлення про помилку
Пов’язана вимога до програмного забезпечення	Авторизація користувачів у системі
Пріоритет	Критичний
Модуль	Модуль входу/виходу з системи та відновлення паролю
Автор, рецензент	Волков Ілля

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розгортання даного програмного забезпечення повинно бути виконане за допомогою Jenkins CI та Amazon Web Services (AWS).

4.1.1 Розгортання серверної частини.

Розгортання серверної частини має складатися з наведених нижче кроків.

Крок 1. Програмний код має бути завантажений з системи контролю версій (Github) за допомогою Jenkins.

Крок 2. Цей програмний код має бути зібраний в ізольований Docker-контейнер за допомогою Jenkins.

Крок 3. Цей контейнер має бути протестований за допомогою Jenkins шляхом запуску автоматичних тестів, написаних під час розробки програми.

Крок 4. Цей контейнер має бути завантажений на сервіс AWS ECS.

Крок 5. У сервісі AWS ECS налаштовуємо роботу одного кластера таким чином, щоб він запускав наш контейнер. Характеристики машин кластеру вказуємо наступну:

Тип процесору: AWS Graviton.

Об‘єм ОЗП: 8 Мб.

Тип: AWS ECS t2.large.

Кількість машин у кластері налаштовуємо згідно наших поточних потреб. В подальшому її можна буде змінювати.

Крок 6. У сервісі AWS CloudFront налаштовуємо перенаправлення на кластер AWS ECS в разі, якщо користувач надсилає запит до серверної частини.

4.1.2 Розгортання клієнтської частини.

Розгортання клієнтської частини має складатися з наведених нижче кроків.

Крок 1. Програмний код має бути завантажений з системи контролю версій (Github) за допомогою Jenkins.

Крок 2. Цей програмний код має бути зібраний в ізольований Docker-контейнер за допомогою Jenkins.

Крок 3. Цей контейнер має бути протестований за допомогою Jenkins шляхом запуску автоматичних тестів, написаних під час розробки програми.

Крок 4. Контейнер має бути запущений. Результатом його запуску має стати завантаження скомпільованого коду клієнтської частини веб-застосування на сервіс AWS S3.

Крок 5. У сервісі AWS CloudFront налаштовуємо перенаправлення на AWS S3 в разі, якщо користувач надсилає запит на отримання веб-сторінки клієнтської частини.

Рисунок 4.1 - Діаграма розгортування

4.2 Робота з програмним забезпеченням

Опис процесу роботи з програмним забезпеченням буде представлений у вигляді інструкції користувача та інструкції адміністратора.

4.2.1 Інструкція користувача.

Наведено в документі «Керівництво користувача».

ВИСНОВКИ

Під час розглядання теоретичного матеріалу з теми даної дипломної роботи були детально розглянуті проблеми навчання сучасних програмістів, був досліджений процес менторингу програмістів, були виявлені його основні переваги та недоліки.

Протягом проектування даного програмного забезпечення були розроблені серверний додаток та клієнтський додаток для даного веб-застосування, а також спроектована база даних.

На етапі тестування були виявлені всі виключні ситуації, які можуть призвести до некоректної роботи програми. На основі отриманих результатів в програмі були передбачені обробки даних ситуацій.

Результатом даного дипломного проекту є веб-застосунок для пошуку менторів і помічників з програмування, який можна використовувати як альтернативу найбільш популярним способам самоосвіти. Він дозволяє Зручно і швидко знаходити собі потрібного та надійного ментора для ефективного навчання..

ПЕРЕЛІК ПОСИЛАНЬ

1) Django [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://ru.wikipedia.org/wiki/Django.

2) Коротко про Django REST Framework [Електронний ресурс]: (Стаття) / Django.fun. – Електрон. дан. (1 файл). – 2018. – Режим доступу: http://django.fun/tutorials/kratko-o-django-rest-framework.

3) AngularJS [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://ru.wikipedia.org/wiki/AngularJS.

4) Model View Controller [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://ru.wikipedia.org/wiki/Model-View-Controller.

5) Dependency injection [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://en.wikipedia.org/wiki/Dependency_injection.

6) Що таке менторство, менторинг [Електронний ресурс]: (Стаття) / Fingeniy. – Електрон. дан. (1 файл). – 2018. – Режим доступу: http://fingeniy.com/chto-takoe-mentorstvo-mentoring.

7) Розвиток і навчання в IT-компаніях [Електронний ресурс]: (Стаття) / Dev. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://dev.by/news/razvitie-i-obuchenie-v-it-kompaniyah.

8) Менторинг в стартапі [Електронний ресурс]: (Стаття) / VC.ru. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://vc.ru/flood/40866-mentoring-v-startape-kogda-ego-vnedryat-i-kak-izvlech-maksimum-polzy.

9) Getting started with AWS ECS [Електронний ресурс]: (Стаття) / Docs.AWS. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://docs.aws.amazon.com/en_us/AmazonECS/latest/developerguide/ECS_GetStarted_EC2.html.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

ВЕБ-ЗАСТОСУВАННЯ ПОШУКУ МЕНТОРІВ ТА ПОМІЧНИКІВ З ПРОГРАМУВАННЯ

Технічне завдання

КПІ.ІП-5102.045440.03.91

“ПОГОДЖЕНО”

Керівник проекту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____І.А. Волков

Київ – 2019 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосування пошуку менторів та помічників з програмування

Галузь застосування:

Наведене технічне завдання поширюється на розробку веб-застосування [КПІ.ІП-5102.045440.03.91], котра використовується для пошуку менторів з ціллю навчання та покращення професійних навичок, для спілкування з менторами в режимі реального часу та оцінювання їх професійних навичок та призначена для програмістів з різними рівнями навичок в якості практикантів та професійних програмістів в якості менторів.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосування є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації і управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для пошуку менторів з різних технологій програмування, та налагодження співпраці між ментором та практикантом на взаємовигідних умовах. Також вона надає можливість будь-якому спеціалісту навчати та ділитись своїми знаннями і навичками з іншими спеціалістами. Розробка включає в себе систему оцінювання менторів, яка сприятиме успішності найбільш професійних фахівців.

Метою даної розробки є спрощення процесу знаходження професійного ментора для тренування конкретних навичок практикантів шляхом автоматизації і оптимізації процесів пошуку та комунікації між практикантами та менторами, та заохочення менторів до навчання практикантів. Також дана розробка надає можливість молодим спеціалістам здобути та перейняти у менторів базові практичні навички, що надзвичайно необхідні для виходу на ринок ІТ-спеціалістів. З боку менторів, дана розробка надає додаткові можливості для самореалізації висококваліфікованих спеціалістів та свого викладацького потенціалу.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача-ментора:

- авторизація в системі за допомогою логін - форми;
- обмін текстовими повідомленнями з менторами та практикантами;
- створення заявок на виконання замовлень;
- виконання замовлень;
- редагування особистих даних.

4.1.1.2 Для користувача-практиканта:

- авторизація в системі за допомогою логін – форми;
- обмін текстовими повідомленнями з менторами;
- створення замовлень на проведення менторингу;
- прийняття заявок на виконання замовлень;
- оцінювання конкретних навичок менторів.

4.1.1.3 Для адміністратора системи:

- редагування даних у системі за допомогою стандартного веб-інтерфейсу Django;

4.1.2 Розробку виконати на платформі Windows 10 або Linux Ubuntu 18.

4.1.3 Додаткові вимоги:

- не зберігати паролі користувачів у відкритому вигляді;
- забезпечити чітке розділення даних, що може отримувати користувач-практикант та даних що може отримувати користувач-ментор.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2 Обслуговування

Вимоги до обслуговування не пред‘являються.

4.3.3 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не пред‘являються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

Конфігурація сервера:

Тип процесору: AWS Graviton (2 виділених віртуальних ядра).

Об‘єм ОЗП: 8 Мб.

Тип: AWS ECS t2.large.

Примітка: на даному залізі повинно бути запущено 2 сервери: 1 основний і 1 дублюючий.

Конфігурація клієнта:

Будь-яка система із підтримкою Google Chrome 56+, Microsoft Edge 15+, Opera 36+, Firefox 56+ або Safari 10+.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows 10, Windows 8.1) або Unix.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: HTTP-запити на сервер з форматом даних типу JSON і методами GET, POST, PUT, DELETE.

4.5.3 Результати повинні бути представлені в наступному форматі: відповіді на HTTP-запити у вигляді JSON, візуальний інтерфейс програми у вікні браузера.

4.5.4 Програмне забезпечення повинно бути створеним на мові Python 3.6 із використанням Django, Django REST Framework (серверна частина) та Typescript із використанням Angular 7 (клієнтська частина).

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему

5.3 У склад супроводжувальної документації повинні входити наведені нижче документи.

5.3.1 Пояснювальна записка не менше ніж на 80 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.3.4 Керівництво адміністратора.

5.3.5 Програма та методика тестування.

5.4 Графічна частина повинна бути виконана на аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Схема структурна компонентів структур даних

5.4.2 Схема структурна класів програмного забезпечення

5.4.3 Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	10.05.2019	
2.	Розробка технічного завдання	19.05.2019	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	16.05.2019	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	18.05.2019	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	24.05.2019	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	24.05.2019	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	26.05.2019	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	26.05.2019	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	26.05.2019	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

ВЕБ-ЗАСТОСУВАННЯ ПОШУКУ МЕНТОРІВ ТА ПОМІЧНИКІВ З ПРОГРАМУВАННЯ

Опис програми

КПІ.ІП-5102.045440.04.13

“ПОГОДЖЕНО”

Керівник проекту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____І.А. Волков

Київ – 2019 року

Тексти програмного коду

Веб-застосування пошуку менторів та помічників з програмування

(Найменування програми (документа))

DVD-R

(Вид носія даних)

15 арк, 112 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2019

1 КОД СЕРВЕРНОЇ ЧАСТИНИ

```
from django.apps import AppConfig


class ApiConfig(AppConfig):
    name = 'api'
from django.conf.urls import url, include

app_name = 'api'

urlpatterns = [
    url(r'^auth/', include('api.auth.urls', namespace='auth')),
    url(r'^users/', include('api.users.urls', namespace='users')),
    url(r'^mentors/', include('api.mentors.urls', namespace='mentors')),
]default_app_config = 'api.apps.ApiConfig'
from django.conf.urls import url
from djoser import views as djoser_views
from rest_framework_simplejwt import views as jwt_views
from api.auth import views as auth_views

app_name = 'api.auth'

urlpatterns = [
    url(r'^login/$', jwt_views.TokenObtainPairView.as_view(), name='user-
login'),
    url(r'^login/refresh/$', jwt_views.TokenRefreshView.as_view(), name='user-
login-refresh'),
    url(r'^logout/$', auth_views.UserLogoutAllView.as_view(), name='user-
logout'),
    url(r'^password/?$', djoser_views.SetPasswordView.as_view(),
name='set_password'),
    url(r'^password/reset/?$', djoser_views.PasswordResetView.as_view(),
name='password_reset'),
    url(r'^password/reset/confirm/?$',
djoser_views.PasswordResetConfirmView.as_view(), name='password_reset_confirm'),
]
import uuid
from rest_framework import views, permissions, status
from rest_framework.response import Response


class UserLogoutAllView(views.APIView):
    """
    Use this endpoint to log out all sessions for a given user.
    """
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request, *args, **kwargs):
        user = request.user
        user.jwt_secret = uuid.uuid4()
        user.save()
        return Response(status=status.HTTP_204_NO_CONTENT)from
utils.helper.fields_handler import (
    FilterOrderManager,
    OrderFilter,
    FilterOrder
)
from utils.mixins.view_mixins import SearcherAndOrderQuerySetMixin
```

```
class MentorsFilterOrderMixin(SearcherAndOrderQuerySetMixin):
    searchers = (
        FilterOrder(query_param='search', filter_keywords=(
            'user__first_name',
            'user__last_name',), use_or=True),
    )

    fom = FilterOrderManager(searchers, ())
from rest_framework import serializers

from users.models import User, Mentor, Address, Review


class AddressSerializer(serializers.ModelSerializer):
    class Meta:
        model = Address
        fields = '__all__'


class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('id',
                  'email',
                  'first_name',
                  'last_name',
                  'login',
                  'is_active',
                  'is_mentor',
                  'created_date',
                  'modified_date',)


class ReviewSerializer(serializers.ModelSerializer):
    author = UserSerializer()

    class Meta:
        model = Review
        fields = ('id',
                  'author',
                  'rank',
                  'comment',
                  'created_date',
                  'modified_date',)


class MentorSerializer(serializers.ModelSerializer):
    user = UserSerializer()
    address = AddressSerializer()
    reviews = ReviewSerializer(many=True)

    class Meta:
        model = Mentor
        fields = ('id',
                  'user',
                  'phone_number',
                  'address',
                  'about',
                  'avatar',
```

```
                  'reviews',
                  'created_date',
                  'modified_date',)


class MentorCreateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Mentor
        fields = ('id',
                  'user',
                  'phone_number',
                  'about',
                  'avatar',
                  'created_date',
                  'modified_date',)
from django.conf.urls import url, include
from . import views

app_name = 'api.mentors'

urlpatterns = [
    url(r'^$', views.MentorsView.as_view(), name='list'),
    url(r'^me/?$', views.SelfMentorView.as_view(), name='mentor'),
    url(r'^upgrade/?$', views.SelfMentorCreateView.as_view(), name='mentor'),
    url(r'^(?P<mentor_uuid>[-a-z0-9]+)/', views.MentorDetailView.as_view(),
name='detail'),
]
from django.db import models
from rest_framework import status
from rest_framework.generics import (
    CreateAPIView,
    DestroyAPIView,
    GenericAPIView,
    ListAPIView,
    ListCreateAPIView,
    RetrieveAPIView,
    RetrieveUpdateDestroyAPIView,
    RetrieveUpdateAPIView,
)
from rest_framework.parsers import JSONParser

from .mixins import MentorsFilterOrderMixin
from .serializers import MentorSerializer, MentorCreateSerializer
from users.models import Mentor, User
from utils.serializers import SerializerPerActionAPIViewMixin
from utils.mixins.auth_mixins import AuthMixin, MentorAuthMixin
from utils.parsers import NestedMultipartParser


class MentorsQuerySetMixin:
    def get_queryset(self):
        queryset = Mentor.objects.filter()
        prefetch_queryset = User.objects.all()
        user_prefetch = models.Prefetch('user',
                                        queryset=prefetch_queryset)
        return queryset.prefetch_related(user_prefetch)


class MentorsView(MentorsFilterOrderMixin,
                  MentorsQuerySetMixin,
                  SerializerPerActionAPIViewMixin,
```

```
                    AuthMixin,
                    ListAPIView):
    """
    Create/get list of mentors
    """
    parser_classes = (JSONParser, NestedMultipartParser,)
    serializer_class = MentorSerializer


class MentorDetailView(MentorsFilterOrderMixin,
                       MentorsQuerySetMixin,
                       SerializerPerActionAPIViewMixin,
                       AuthMixin,
                       RetrieveUpdateAPIView):
    """
    Create/get list of mentors
    """
    parser_classes = (JSONParser, NestedMultipartParser,)
    serializer_class = MentorSerializer
    lookup_url_kwarg = 'mentor_uuid'


class SelfMentorView(MentorAuthMixin,
                     RetrieveUpdateAPIView):
    """
    Retrieve/update mentor.
    """
    queryset = Mentor.objects.all()
    parser_classes = (JSONParser, NestedMultipartParser, )
    serializer_class = MentorSerializer

    def get_object(self, *args, **kwargs):
        return self.queryset.get(user=self.request.user)


class SelfMentorCreateView(AuthMixin,
                           CreateAPIView):
    """
    Retrieve/update mentor.
    """
    queryset = Mentor.objects.all()
    parser_classes = (JSONParser, NestedMultipartParser, )
    serializer_class = MentorCreateSerializer

    def create(self, request, *args, **kwargs):
        request.data["user"] = self.request.user.id
        resp = super().create(request, *args, **kwargs)
        if resp.status_code == status.HTTP_201_CREATED:
            User.objects.filter(id=self.request.user.id).update(is_mentor=True)
        return resp
from utils.helper.fields_handler import (
    FilterOrderManager,
    OrderFilter,
    FilterOrder
)
from utils.mixins.view_mixins import SearcherAndOrderQuerySetMixin


class MentorsFilterOrderMixin(SearcherAndOrderQuerySetMixin):
    searchers = (
        FilterOrder(query_param='search', filter_keywords=(
```

```
            'user__first_name',
            'user__last_name',), use_or=True),
    )

    fom = FilterOrderManager(searchers, ())
from rest_framework import serializers

from users.models import User, Mentor, Address, Review


class AddressSerializer(serializers.ModelSerializer):
    class Meta:
        model = Address
        fields = '__all__'


class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('id',
                  'email',
                  'first_name',
                  'last_name',
                  'login',
                  'is_active',
                  'is_mentor',
                  'created_date',
                  'modified_date',)


class ReviewSerializer(serializers.ModelSerializer):
    author = UserSerializer()

    class Meta:
        model = Review
        fields = ('id',
                  'author',
                  'rank',
                  'comment',
                  'created_date',
                  'modified_date',)


class MentorSerializer(serializers.ModelSerializer):
    user = UserSerializer()
    address = AddressSerializer()
    reviews = ReviewSerializer(many=True)

    class Meta:
        model = Mentor
        fields = ('id',
                  'user',
                  'phone_number',
                  'address',
                  'about',
                  'avatar',
                  'reviews',
                  'created_date',
                  'modified_date',)
```

```
class MentorCreateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Mentor
        fields = ('id',
                  'user',
                  'phone_number',
                  'about',
                  'avatar',
                  'created_date',
                  'modified_date',)
from django.conf.urls import url, include
from . import views

app_name = 'api.orders'

urlpatterns = [
    url(r'^$', views.MentorsView.as_view(), name='list'),
    url(r'^me/?$', views.SelfMentorView.as_view(), name='mentor'),
    url(r'^upgrade/?$', views.SelfMentorCreateView.as_view(), name='mentor'),
    url(r'^(?P<mentor_uuid>[-a-z0-9]+)/', views.MentorDetailView.as_view(),
name='detail'),
]
from django.db import models
from rest_framework import status
from rest_framework.generics import (
    CreateAPIView,
    DestroyAPIView,
    GenericAPIView,
    ListAPIView,
    ListCreateAPIView,
    RetrieveAPIView,
    RetrieveUpdateDestroyAPIView,
    RetrieveUpdateAPIView,
)
from rest_framework.parsers import JSONParser

from .mixins import MentorsFilterOrderMixin
from .serializers import MentorSerializer, MentorCreateSerializer
from users.models import Mentor, User
from utils.serializers import SerializerPerActionAPIViewMixin
from utils.mixins.auth_mixins import AuthMixin, MentorAuthMixin
from utils.parsers import NestedMultipartParser


class MentorsQuerySetMixin:
    def get_queryset(self):
        queryset = Mentor.objects.filter()
        prefetch_queryset = User.objects.all()
        user_prefetch = models.Prefetch('user',
                                        queryset=prefetch_queryset)
        return queryset.prefetch_related(user_prefetch)


class MentorsView(MentorsFilterOrderMixin,
                  MentorsQuerySetMixin,
                  SerializerPerActionAPIViewMixin,
                  AuthMixin,
                  ListAPIView):
    """
    Create/get list of mentors
    """
```

```
    parser_classes = (JSONParser, NestedMultipartParser,)
    serializer_class = MentorSerializer


class MentorDetailView(MentorsFilterOrderMixin,
                       MentorsQuerySetMixin,
                       SerializerPerActionAPIViewMixin,
                       AuthMixin,
                       RetrieveUpdateAPIView):
    """
    Create/get list of mentors
    """
    parser_classes = (JSONParser, NestedMultipartParser,)
    serializer_class = MentorSerializer
    lookup_url_kwarg = 'mentor_uuid'


class SelfMentorView(MentorAuthMixin,
                     RetrieveUpdateAPIView):
    """
    Retrieve/update mentor.
    """
    queryset = Mentor.objects.all()
    parser_classes = (JSONParser, NestedMultipartParser, )
    serializer_class = MentorSerializer

    def get_object(self, *args, **kwargs):
        return self.queryset.get(user=self.request.user)


class SelfMentorCreateView(AuthMixin,
                           CreateAPIView):
    """
    Retrieve/update mentor.
    """
    queryset = Mentor.objects.all()
    parser_classes = (JSONParser, NestedMultipartParser, )
    serializer_class = MentorCreateSerializer

    def create(self, request, *args, **kwargs):
        request.data["user"] = self.request.user.id
        resp = super().create(request, *args, **kwargs)
        if resp.status_code == status.HTTP_201_CREATED:
            User.objects.filter(id=self.request.user.id).update(is_mentor=True)
        return resp
from django.conf.urls import url, include
from djoser import views as djoser_views

app_name = 'api.users'

urlpatterns = [
    url(r'^me/?$', djoser_views.UserView.as_view(), name='user'),
    url(r'^create/?$', djoser_views.UserCreateView.as_view(), name='user-
create'),
    url(r'^delete/?$', djoser_views.UserDeleteView.as_view(), name='user-
delete'),
]
```

2 КОД КЛІЄНТСЬКОЇ ЧАСТИНИ

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router/src/directives/router_outlet';
import { ActivatedRoute } from '@angular/router';

import { crossfadeAnimation } from './animations/fade-in-out.animation';
import { LoginService } from './services/login.service';
import { RegisterIconsService } from './services/register-icons.service';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.scss'],
    animations: [crossfadeAnimation]
})
export class AppComponent {
    public constructor(private loginService: LoginService,
                       private registerIconsService: RegisterIconsService) {
        this.registerIconsService.registerDefaultIcons();
    }

    public getRouterOutletState(outlet: RouterOutlet): ActivatedRoute {
        return outlet.isActivated ? outlet.activatedRoute : null;
    }
}
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import {
BsDropdownModule, ButtonsModule, ModalModule, PaginationModule,
ProgressbarModule, TabsModule, TimepickerModule,
TooltipModule
} from 'ngx-bootstrap';
import { NgxWebstorageModule } from 'ngx-webstorage';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app.routing.module';
import { APP_PROVIDERS } from './services';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { APP_GUARDS } from './guards';
import { ToastModule } from './components/shared/toast-module/toast.module';

@NgModule({
    declarations: [
        AppComponent
    ],
    imports: [
        CommonModule,
        ReactiveFormsModule,
        AppRoutingModule,
        BrowserModule,
        FormsModule,
        HttpClientModule,
        BrowserAnimationsModule,
        ToastModule.forRoot(),
        BsDropdownModule.forRoot(),
```

```
        ButtonsModule.forRoot(),
        TooltipModule.forRoot(),
        TabsModule.forRoot(),
        ModalModule.forRoot(),
        ProgressbarModule.forRoot(),
        TimepickerModule.forRoot(),
        PaginationModule.forRoot(),
        NgxWebstorageModule.forRoot({ prefix: 'app', separator: '.' })
    ],
    providers: [
        ...APP_PROVIDERS,
        ...APP_GUARDS
    ],
    bootstrap: [AppComponent]
})
export class AppModule {
}
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { AuthGuard, LoginPageGuard } from './guards';

export const routes: Routes = [
    {
        path: '',
        redirectTo: 'login',
        pathMatch: 'full'
    },
    {
        path: 'login',
        loadChildren: './components/login/login.module#LoginModule',
        canLoad: [LoginPageGuard],
        canActivate: [LoginPageGuard]
    },
    {
        path: 'password',
        loadChildren: './components/reset-password/reset-
password.module#ResetPasswordModule'
    },
    {
        path: 'main',
        loadChildren: './components/main/main.module#MainModule',
        canActivate: [AuthGuard],
        canActivateChild: [AuthGuard]
    },
    {
        path: '**',
        redirectTo: 'login'
    }
];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule {
}
import { animate, animateChild, group, query, style, transition, trigger } from
'@angular/animations';

export const fadeInOutAnimation =
```

```
    trigger('fadeInOutAnimation', [

        transition(':enter', [

            style({ opacity: 0 }),

            animate('.3s', style({ opacity: 1 }))
        ]),
        transition(':leave', [

            style({ opacity: 1 }),

            animate('.3s', style({ opacity: 0 }))
        ])
    ]);

export const fadeInAnimation =
    trigger('fadeInAnimation', [

        // route 'enter' transition
        transition(':enter', [

            // css styles at start of transition
            style({ opacity: 0 }),

            // animation and styles at end of transition
            animate('10s', style({ opacity: 1 }))
        ])
    ]);

export const fadeOutAnimation =

    trigger('fadeOutAnimation', [

        transition('* => *', [

            query(':enter',
                [
                    style({ opacity: 0 })
                ],
                { optional: true }
            ),

            query(':leave',
                [
                    style({ opacity: 1 }),
                    animate('1s', style({ opacity: 0 })),
                    animateChild()
                ],
                { optional: true }
            ),

            query(':enter',
                [
                    style({ opacity: 0 }),
                    animate('1s', style({ opacity: 1 })),
                    animateChild()
                ],
                { optional: true }
            )
```

```
        ])

    ]);

export const crossfadeAnimation =

    trigger('crossfadeAnimation', [

        transition('* => *', [

            query(':enter',
                [
                    style({ opacity: 0 })
                ],
                { optional: true }
            ),

            group([
                query(':leave',
                    [
                        style({ opacity: 1 }),
                        animate('1s', style({ opacity: 0 })),
                        animateChild()
                    ],
                    { optional: true }
                ),

                query(':enter',
                    [
                        style({ opacity: 0 }),
                        animate('1s', style({ opacity: 1 })),
                        animateChild()
                    ],
                    { optional: true }
                )
            ])

        ])

    ]);
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login.component';
import { LoginFormComponent } from './login-form/login-form.component';
import { ForgotFormComponent } from './forgot-form/forgot-form.component';
import { RegisterFormComponent } from './register-form/register-form.component';

export const routes: Routes = [
    {
        path: '', component: LoginComponent, data: { title: getTitle },
        children: [
            { path: '', component: LoginFormComponent, pathMatch: 'full' },
            { path: 'register', component: RegisterFormComponent },
            { path: 'forgot', component: ForgotFormComponent }
        ]
    }
];

@NgModule({
    imports: [RouterModule.forChild(routes)],
    exports: [RouterModule]
```

```
})
export class LoginRoutingModule {
}

export function getTitle(): string {
    return 'Login';
}
import { Component } from '@angular/core';

@Component({
    selector: 'app-login',
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.scss']
})
export class LoginComponent {}
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ReactiveFormsModule } from '@angular/forms';

import { LoginFormComponent } from './login-form/login-form.component';
import { ForgotFormComponent } from './forgot-form/forgot-form.component';
import { LoginComponent } from './login.component';
import { LoginRoutingModule } from './login-routing.module';
import { AppFormsModule } from '../shared/app-forms-module/app-forms.module';
import { PortalInputComponent } from './portal-input/portal-input.component';
import { SharedComponentsModule } from '../shared';
import { RegisterFormComponent } from './register-form/register-form.component';

@NgModule({
    imports: [
        CommonModule,
        LoginRoutingModule,
        AppFormsModule,
        SharedComponentsModule,
        ReactiveFormsModule
    ],
    declarations: [
        LoginComponent,
        LoginFormComponent,
        RegisterFormComponent,
        ForgotFormComponent,
        PortalInputComponent
    ]
})
export class LoginModule {}
import { Component, OnInit } from '@angular/core';
import { FormControl, Validators } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { LoginService } from '../../../services/login.service';
import { ToastService } from '../../../components/shared/toast-
module/toast.service';
import { CredentialsValidators } from
'../../../utils/validators/credentials.validators';
import { FormGroupWrapper } from '../../shared/app-forms-module/form-group/form-
group-wrapper';

@Component({
    selector: 'app-forgot-form',
    templateUrl: './forgot-form.component.html',
    styleUrls: ['./forgot-form.component.scss']
})
```

```
export class ForgotFormComponent implements OnInit {
    public loginGroup: FormGroupWrapper;
    public email: FormControl;

    constructor(private login: LoginService,
                public toast: ToastService,
                private router: Router,
                private route: ActivatedRoute) {
    }

    public ngOnInit(): void {
        this.email = new FormControl('', Validators.compose([
            Validators.required,
            Validators.minLength(2),
            Validators.maxLength(54),
            CredentialsValidators.emailValidation]));
        this.loginGroup = new FormGroupWrapper({
            email: this.email
        });
    }

    public onSubmit(): void {
        const value = {
            email: this.loginGroup.value.email,
            is_admin_url: true
        };
        this.login.resetPassword(value).subscribe(() => {
            this.toast.show('success', 'Email was successfully sent');
        });
    }

    public goToLogin(): void {
        this.router.navigate(['../'], { relativeTo: this.route });
    }
}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

ВЕБ-ЗАСТОСУВАННЯ ПОШУКУ МЕНТОРІВ ТА ПОМІЧНИКІВ З ПРОГРАМУВАННЯ

Програма та методика тестування

КПІ.ІП-5102.045440.05.51

“ПОГОДЖЕНО”

Керівник проекту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____І.А. Волков

Київ – 2019 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-застосування пошуку менторів та помічників з програмування, що представляє із себе клієнт-серверний застосунок, створений за допомогою фреймворку Angular 7 на клієнтській частині та Django, Django REST Framework на серверній частині.

2 МЕТА ТЕСТУВАННЯ

Під час тестування мають бути перевірені всі функціональні і нефункціональні вимоги до даного проекту:

- авторизація менторів в системі за допомогою логін - форми;
- обмін менторами текстовими повідомленнями з менторами та практикантами;
- створення менторами заявок на виконання замовлень;
- виконання менторами замовлень;
- редагування менторами особистих даних.
- авторизація практикантів в системі за допомогою логін – форми;
- обмін практикантами текстовими повідомленнями з менторами;
- створення практикантами замовлень на проведення менторингу;
- прийняття практикантами заявок на виконання замовлень;
- оцінювання практикантами конкретних навичок менторів.
- інтуїтивність візуального інтерфейсу;
- висока стійкість до відмов;
- висока швидкість завантаження та видачі результатів запитів;
- наочне відображення помилок для користувача.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування для проекту PROboost складатиметься з наступних методів:

а) юніт-тестування;

б) інтеграційне тестування;

в) системне тестування

1) функціональне тестування;

2) GUI (Graphic User Interface) тестування;

3) навантажувальне тестування;

г) UAT (User Acceptance Testing).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

а) Юніт-тестування.

Буде автоматизоване за допомогою засобів Django та буде налаштований процес автоматичного запуску тестів під час кожної збірки програми за допомогою Jenkins CI. Виконуватиметься розробником.

б) Інтеграційне тестування.

Буде автоматизоване за допомогою Mocha та буде налаштований процес автоматичного запуску тестів під час кожної збірки програми за допомогою Jenkins CI. Виконуватиметься розробником.

в) Системне тестування.

1) Функціональне тестування.

Виконуватиметься тестувальником після успішного виконання попереднього етапу. В разі невдачі програма буде повернена на доопрацювання. Проводитиметься в ручному режимі.

2) GUI тестування.

Виконуватиметься тестувальником паралельно із попереднім етапом. В разі невдачі програма буде повернена на доопрацювання. Проводитиметься в ручному режимі.

3) Навантажувальне тестування.

Виконуватиметься тестувальником після всіх попередніх етапом. В разі невдачі програма буде повернена на доопрацювання та оптимізацію. Проводитиметься в автоматизованому режимі.

г) UAT.

Виконуватиметься замовником. В разі невдачі повернутися до системного тестування. В разі успіху – визнати проект повністю завершеним та перейти до його релізу. Проводитиметься в ручному режимі.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ О.А. Павлов

“___” _____ 2019 р.

ВЕБ-ЗАСТОСУВАННЯ ПОШУКУ МЕНТОРІВ ТА ПОМІЧНИКІВ З ПРОГРАМУВАННЯ

Керівництво користувача

КПІ.ІП-5102.045440.06.34

“ПОГОДЖЕНО”

Керівник проекту:

_____ К.І. Ліщук

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____І.А. Волков

Київ – 2019 року

В даному документі представлене керівництво користувача для роботи з програмним застосунком даного дипломного проекту.

а) Заходимо на головну сторінку веб-застосування.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.1 - Головна сторінка

б) Натискаємо кнопку “Register” та потрапляємо на форму для реєстрації користувача. Вводимо валідні дані.

Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.2 - Форма реєстрації

в) Ще раз натискаємо кнопку “Register” та потрапляємо на сторінку пошуку менторів. За допомогою панелі зліва ми можемо перейти на сторінку «Мої замовлення». За допомогою поля пошуку ми можемо здійснювати пошук менторів. Натискаючи на менторів в списку ми можемо дізнатись більш детальну інформацію про кожного з ним.

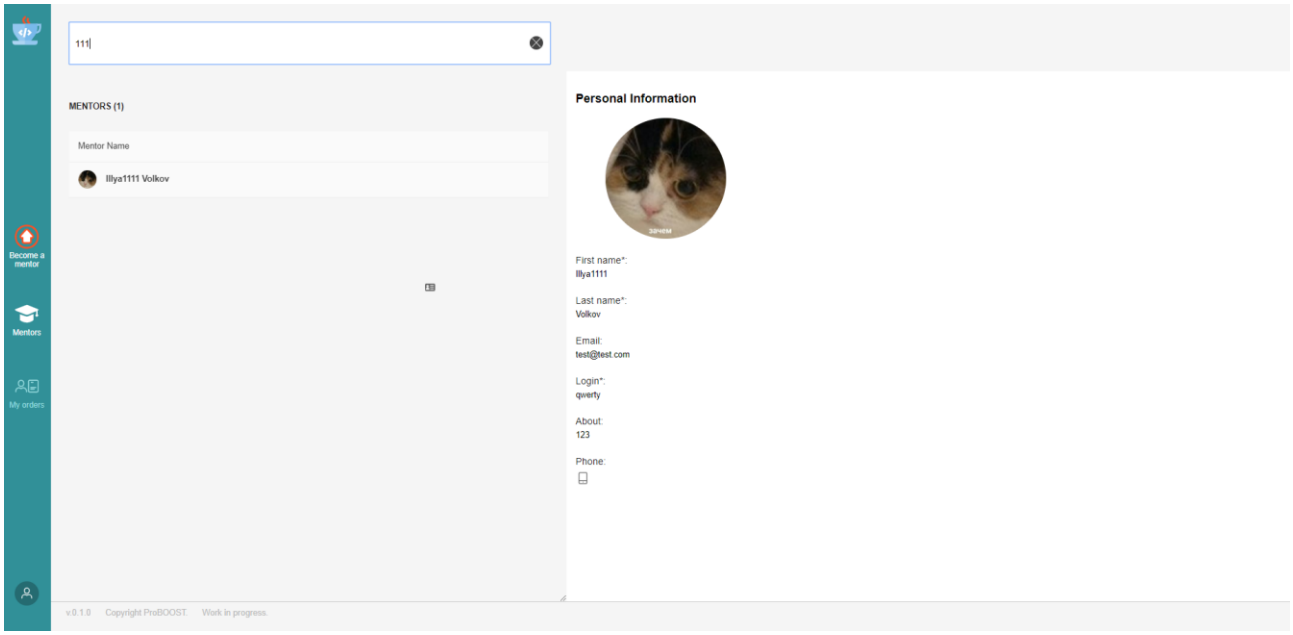


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.3 - Сторінка пошуку менторів

г) Натискаємо значок користувача. Відкривається випадаюче меню. Натискаємо кнопку «My Account» та потрапляємо на сторінку редагування персональних даних. За допомогою полів вводу та кнопки «Save» можемо змінювати дані про себе.

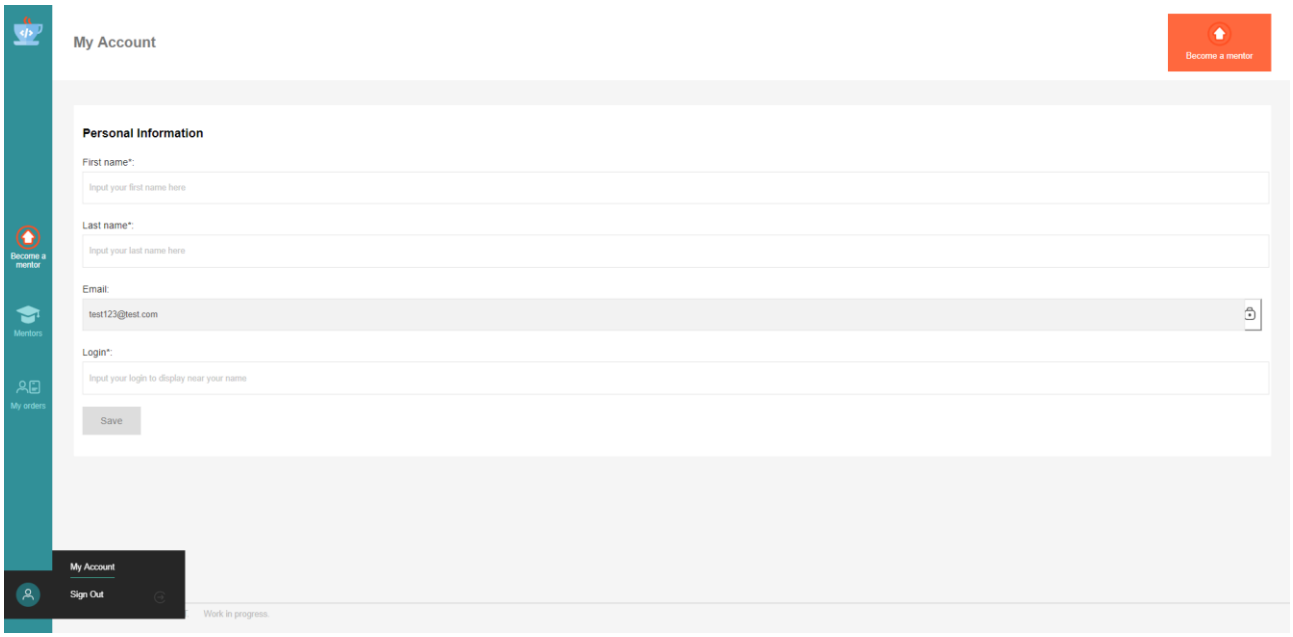


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.4 - Сторінка редагування персональних даних

д) Натискаємо кнопку «Become a Mentor» та потрапляємо на сторінку реєстрації ментора. За допомогою полів вводу та кнопки «Save» можемо задати початкові дані про себе як про ментора.

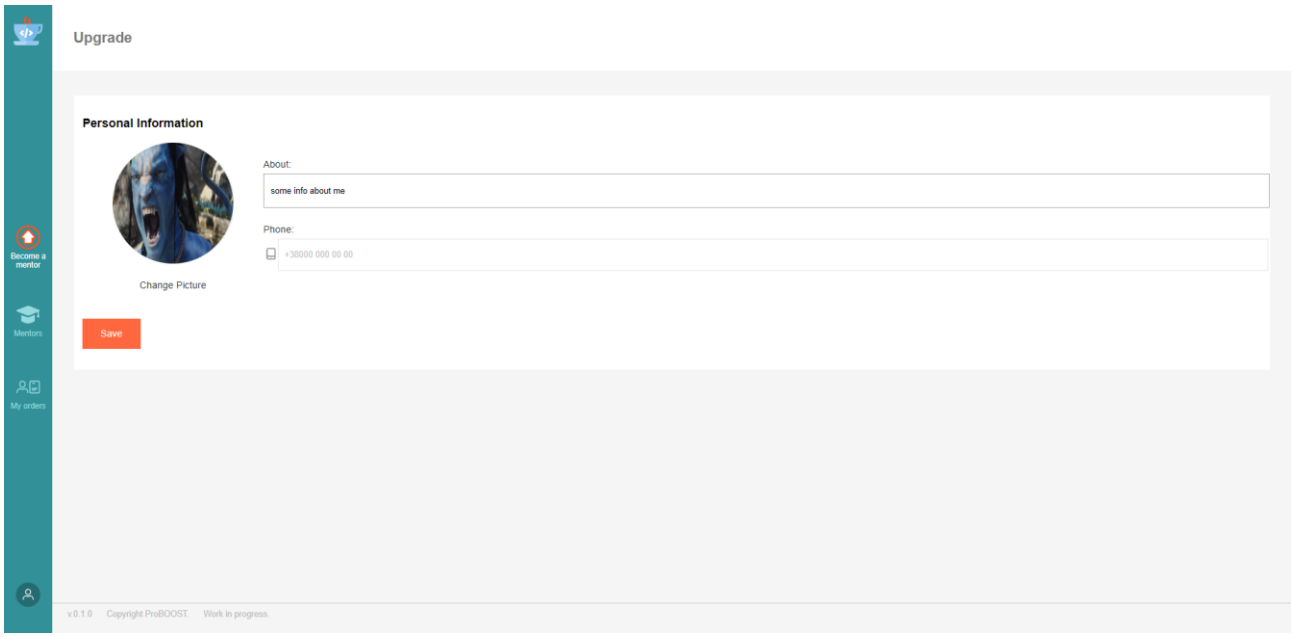


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.5 - Сторінка реєстрації ментора

е) Після збереження автоматично потрапляємо на схожу сторінку редагування персональних даних ментора.

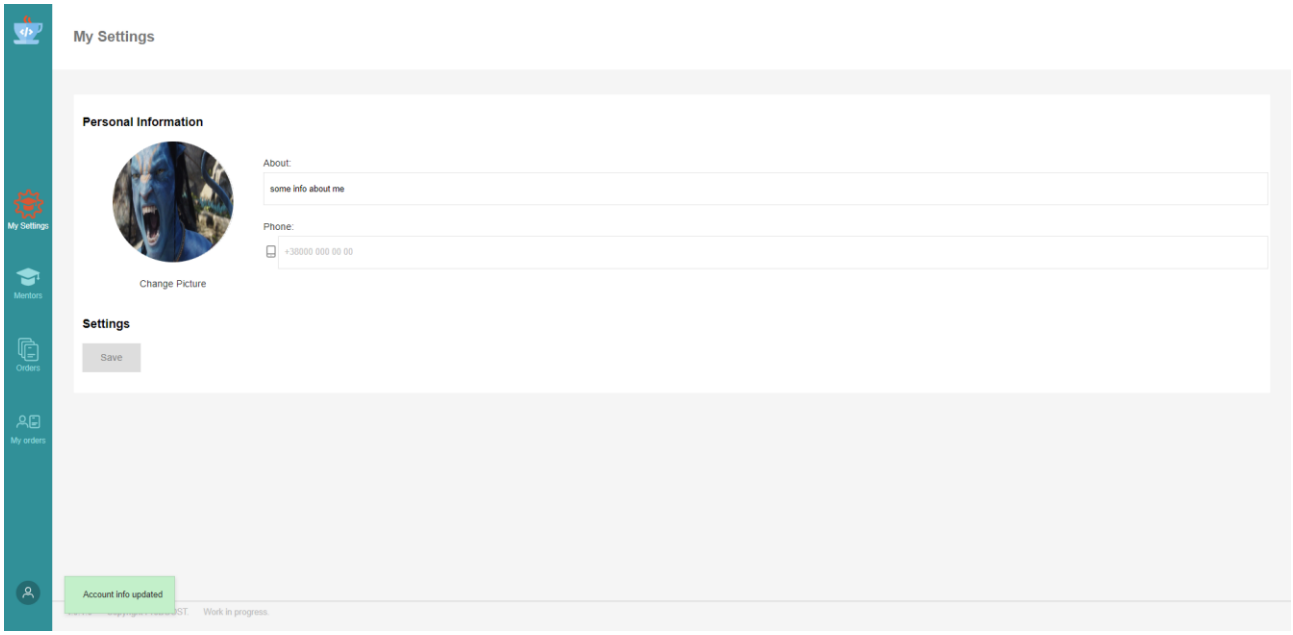


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.6

- Сторінка редагування персональних даних ментора

ж) Для виходу з облікового запису натискаємо значок користувача та кнопку «Sign Out» у випадаючому меню.

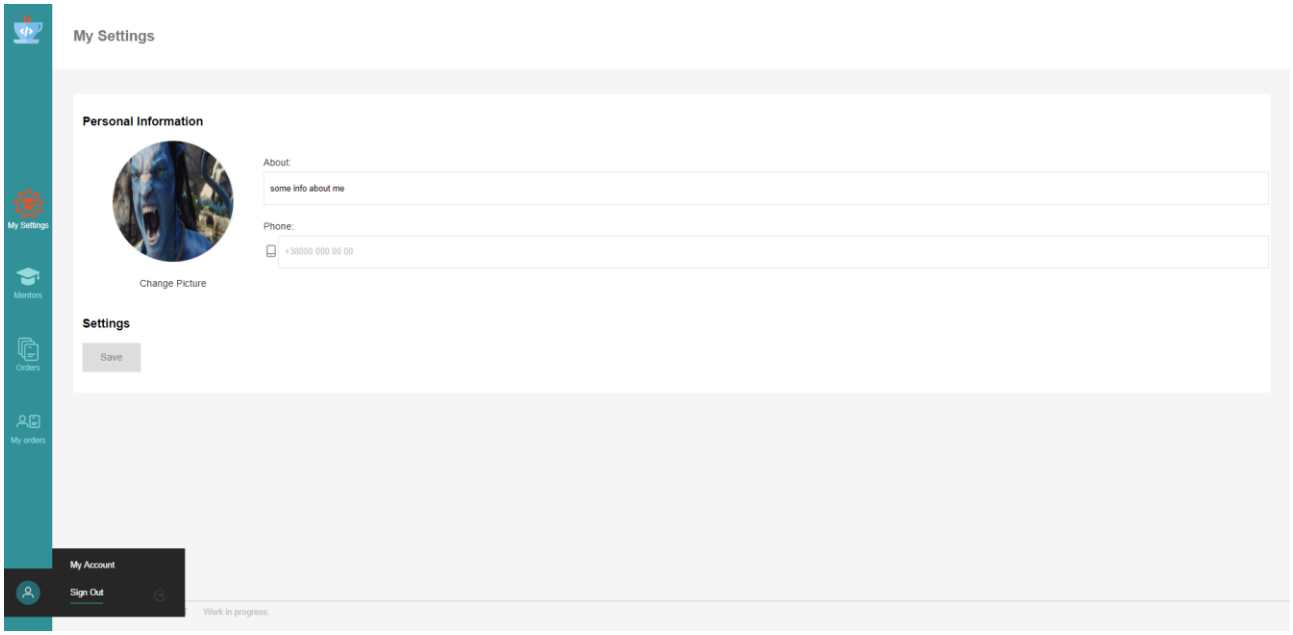


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.7

- Вихід з системи

Інструкція адміністратора.

а) Заходимо на сторінку адміністратора веб-застосування. Вхід здійснюємо за допомогою наданих адміністратору даних.

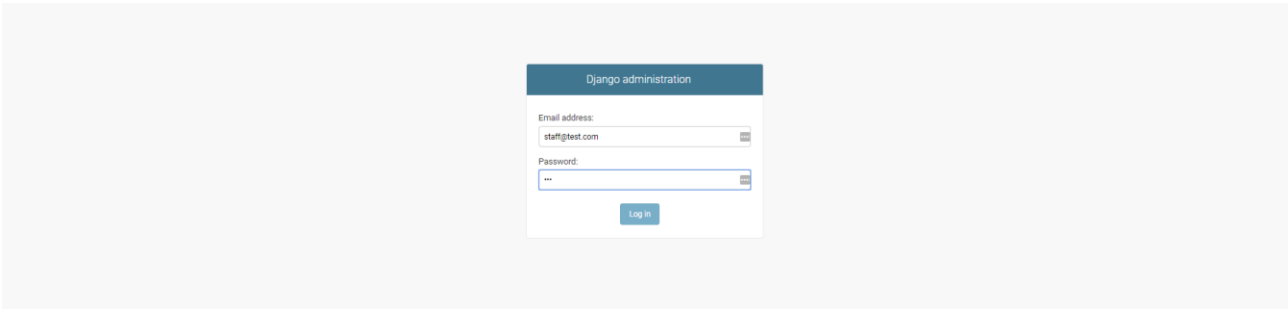


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.8 - Форма входу в додаток адміністрування

б) Потрапляємо на головну сторінку додатку адміністрування. Вибираємо ті записи, які бажаємо проглянути чи редагувати.

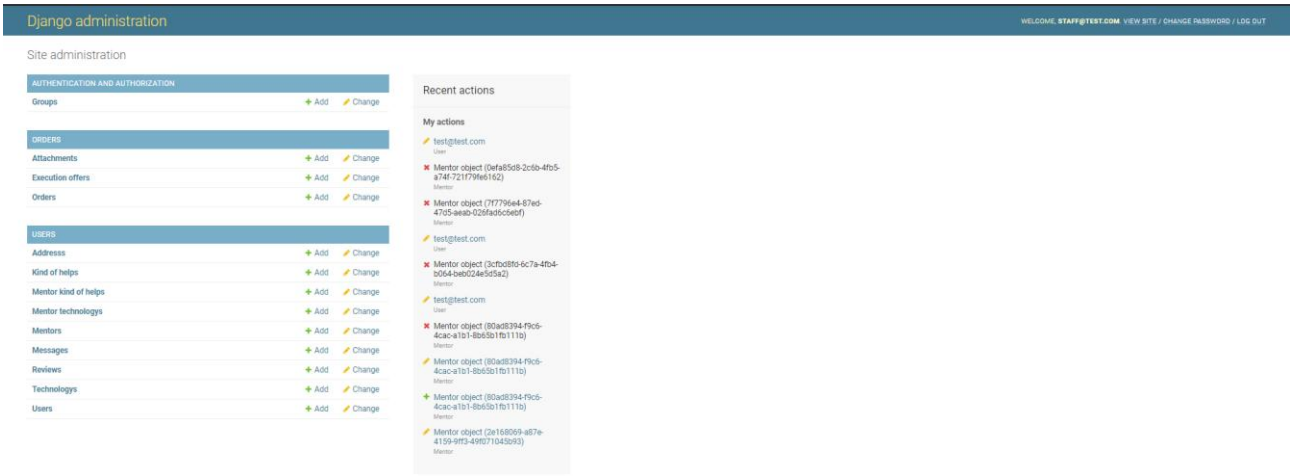


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.9 - Головна сторінка додатку адміністрування

в) Потрапляємо на головну список записів у БД. Вибираємо запис, який нас цікавить.

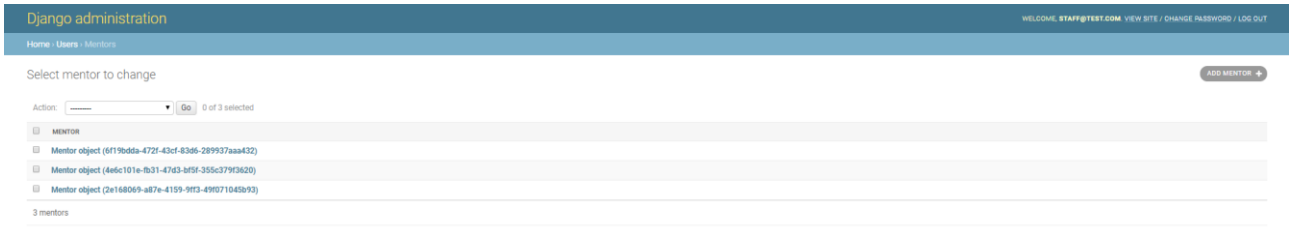


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.10 - Список записів у БД

г) Потрапляємо на сторінку редагування записів у БД. За допомогою полів вводу та кнопки «Save» можемо редагувати конкретний запис без використання нашого веб-застосування.

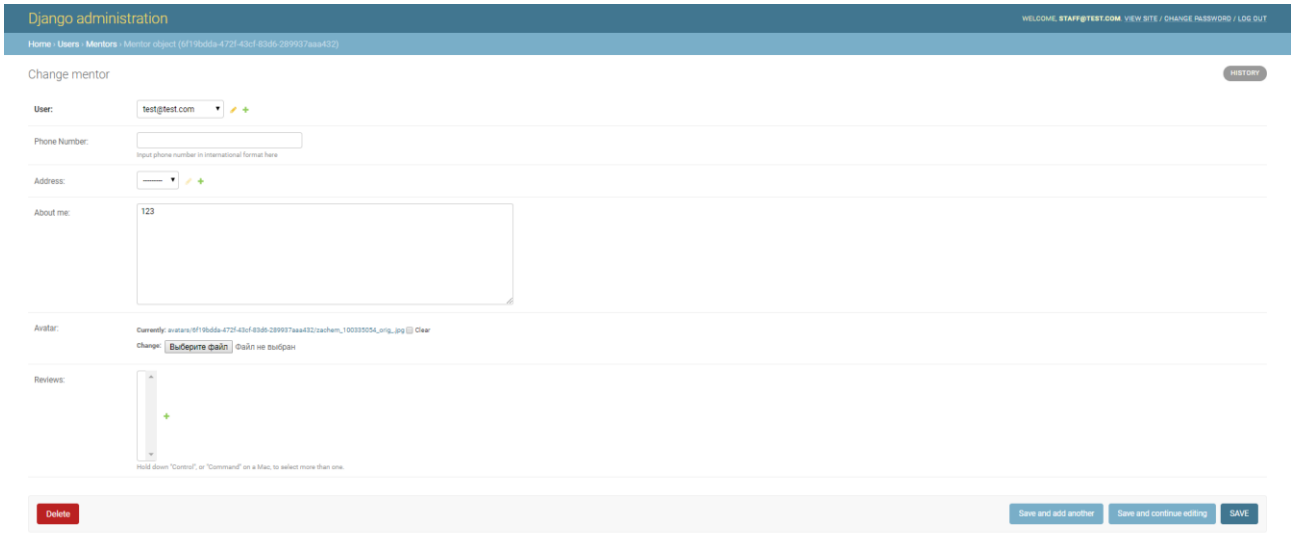
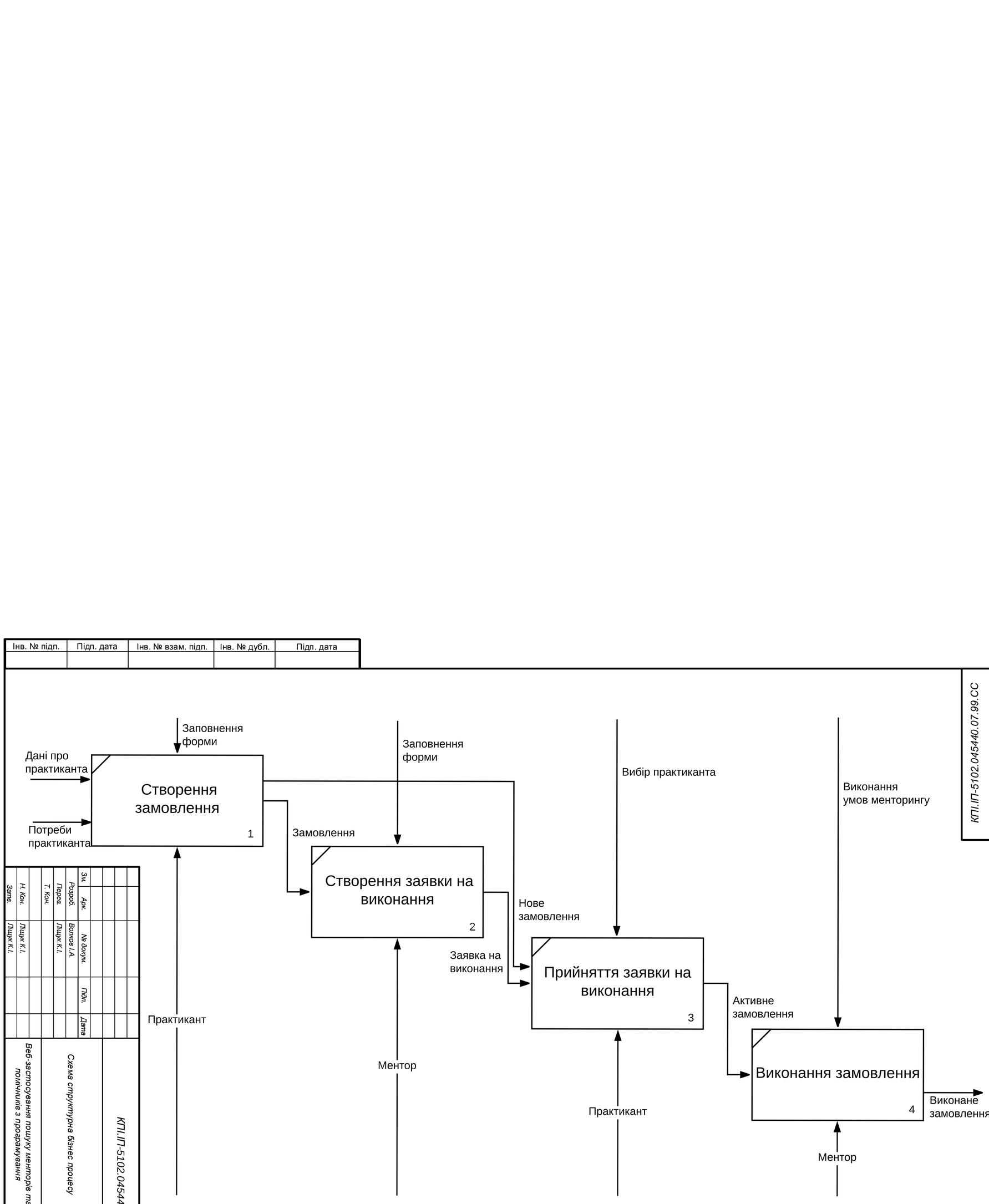


Рисунок **Ошибка! Текст указанного стиля в документе отсутствует.**.11 - Редагування записів у БД

Інв. № підп.	Підп. дата	Інв. № взам. підп.	Інв. № дубл.	Підп. дата

КПІ.ІП-5102.045440.07.99.СС



Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Волков І.А.		
Перев.		Пщук К.І.		
Т. Кон.				
Н. Кон.		Пщук К.І.		
Затв.		Пщук К.І.		

КПІ.ІП-5102.045440.07.99.СС

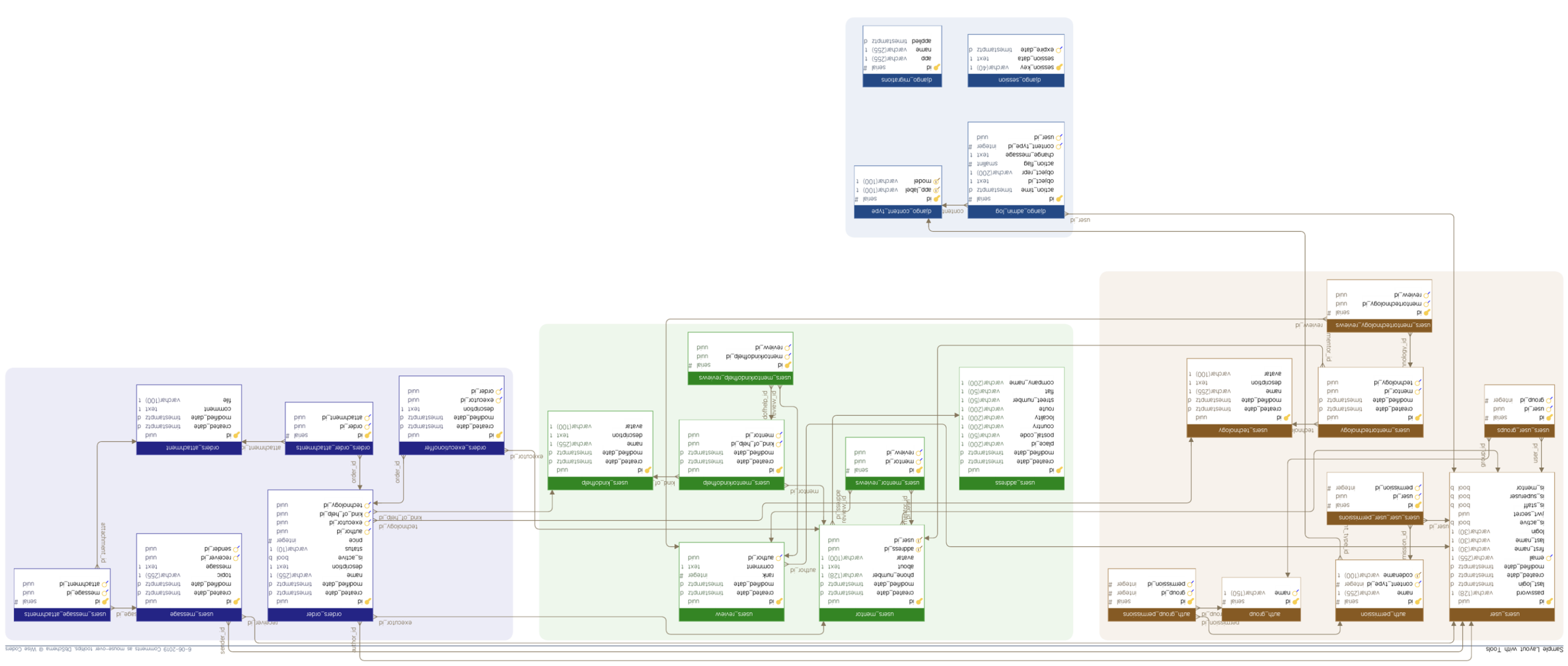
Схема структурна бізнес процесу

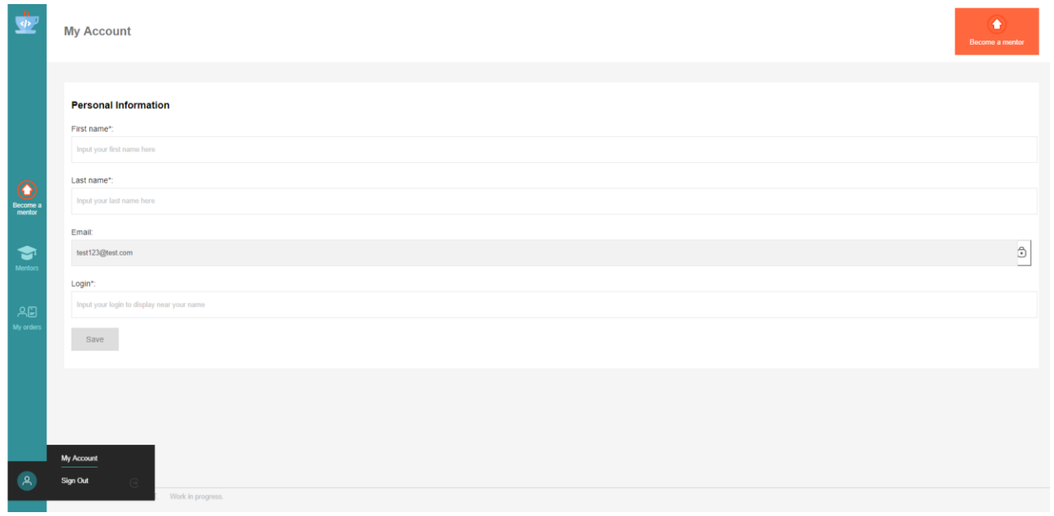
Веб-застосування пошуку менторів та помічників з програмування

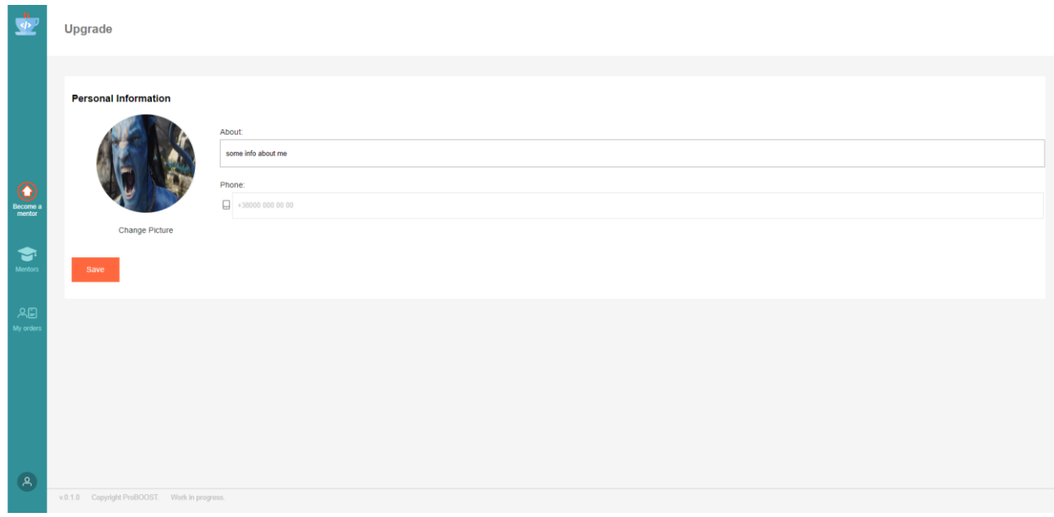
Літ.	Арк.	Аркушів
	1	

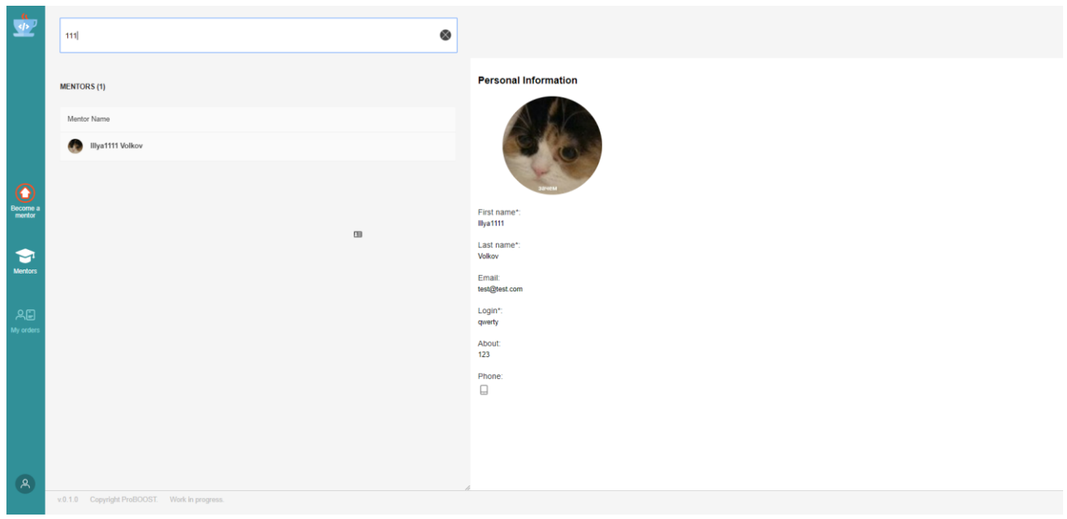
КПІ ім.Ігоря Сікорського
Кафедра АСОІУ
гр. ІП-51

Аркуш

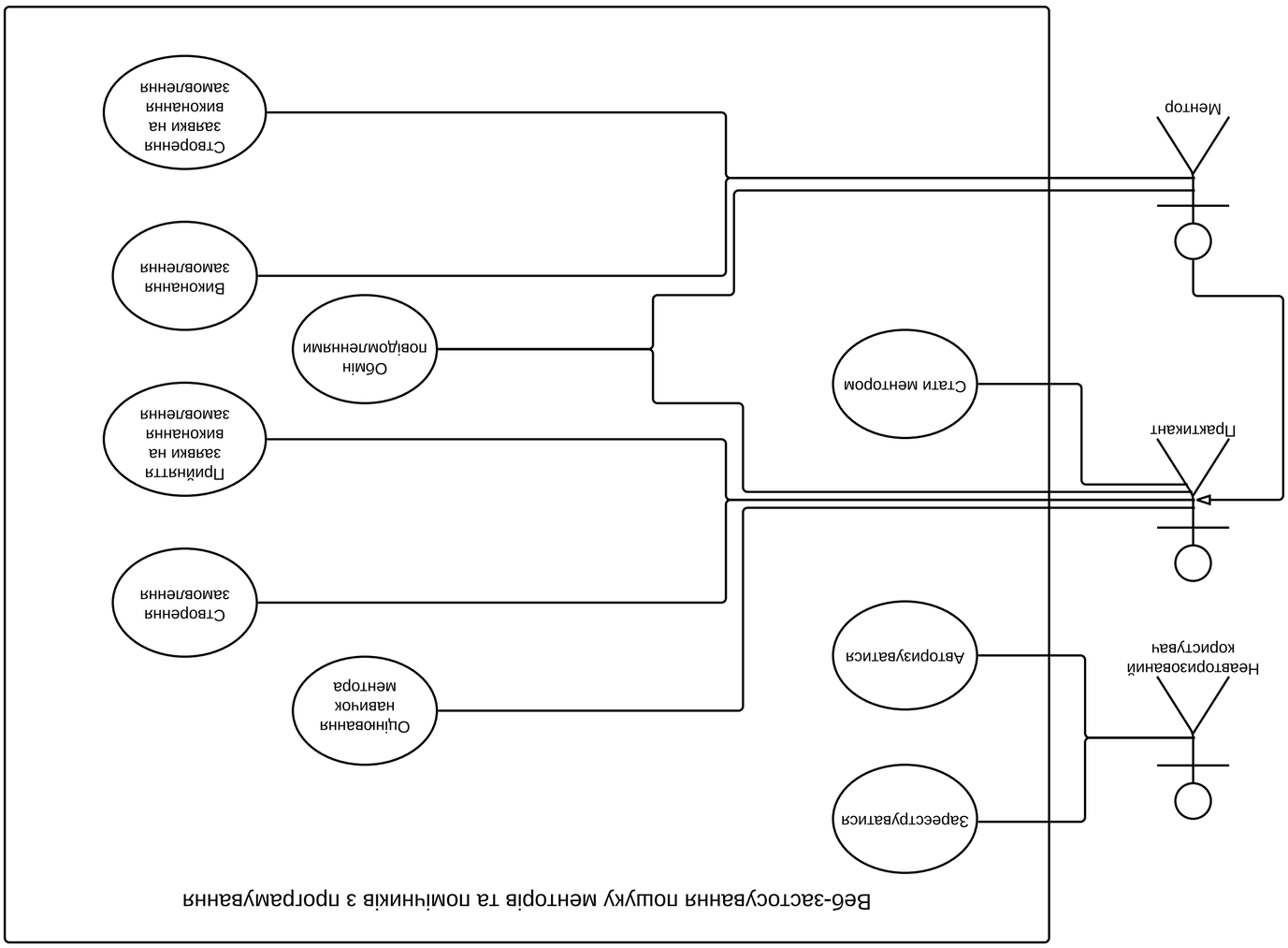

Sample Layout with Tools
6-06-2019 Comments as mouse-over tooltips. DbSchema @ Wise Coders
users_user
id serial #
password varchar(128) 1
last_login timestamptz d
created_date timestamptz d
modified_date timestamptz d
email varchar(255) 1
first_name varchar(30) 1
last_name varchar(30) 1
login varchar(30) 1
pwt_secret uuid
is_active bool b
is_staff bool b
is_superuser bool b
is_mentor bool b
users_user_permissions
id serial #
user_id uuid
permission_id integer #
users_user_groups
id serial #
user_id uuid
group_id integer #
auth_permission
id serial #
name varchar(255) 1
content_type_id integer #
codename varchar(100) 1
auth_group
id serial #
name varchar(150) 1
auth_group_permissions
id serial #
group_id integer #
permission_id integer #
users_mentor
id uuid
created_date timestamptz d
modified_date timestamptz d
phone_number varchar(128) 1
about text 1
avatar varchar(100) 1
address_id uuid
user_id uuid
users_address
id uuid
created_date timestamptz d
modified_date timestamptz d
place_id varchar(200) 1
postal_code varchar(50) 1
locality varchar(200) 1
country varchar(200) 1
route varchar(200) 1
street_number varchar(50) 1
flat varchar(50) 1
company_name varchar(200) 1
users_mentor_reviews
id serial #
mentor_id uuid
review_id uuid
users_mentortechnology_reviews
id serial #
mentortechnology_id uuid
review_id uuid
users_mentortechnology
id uuid
created_date timestamptz d
modified_date timestamptz d
mentor_id uuid
technology_id uuid
users_technology
id uuid
created_date timestamptz d
modified_date timestamptz d
name varchar(255) 1
description text 1
avatar varchar(100) 1
users_mentorkindofhelp
id serial #
mentorkindofhelp_id uuid
review_id uuid
users_mentorkindofhelp_reviews
id serial #
mentorkindofhelp_id uuid
review_id uuid
users_kindofhelp
id uuid
created_date timestamptz d
modified_date timestamptz d
name varchar(255) 1
description text 1
avatar varchar(100) 1
users_review
id uuid
created_date timestamptz d
modified_date timestamptz d
rank integer #
comment text 1
author_id uuid
orders_order
id uuid
created_date timestamptz d
modified_date timestamptz d
name varchar(255) 1
description text 1
is_active bool b
status varchar(10) 1
price integer #
author_id uuid
executor_id uuid
kind_of_help_id uuid
technology_id uuid
orders_executionoffer
id uuid
created_date timestamptz d
modified_date timestamptz d
description text 1
executor_id uuid
order_id uuid
orders_order_attachments
id serial #
order_id uuid
attachment_id uuid
orders_attachment
id uuid
created_date timestamptz d
modified_date timestamptz d
comment text 1
file varchar(100) 1
users_message
id uuid
created_date timestamptz d
modified_date timestamptz d
topic varchar(255) 1
message text 1
receiver_id uuid
sender_id uuid
users_message_attachments
id serial #
message_id uuid
attachment_id uuid
django_admin_log
id serial #
action_time timestamptz d
object_id text 1
object_repr varchar(200) 1
action_flag smallint #
change_message text 1
content_type_id integer #
user_id uuid
django_content_type
id serial #
app_label varchar(100) 1
model varchar(100) 1
django_session
session_key varchar(40) 1
session_data text 1
expire_date timestamptz d
django_migrations
id serial #
app varchar(255) 1
name varchar(255) 1
applied timestamptz d
user_id
content_type_id
permission_id
group_id
author_id
mentor_id
review_id
kind_of_help_id
technology_id
executor_id
order_id
attachment_id
message_id
sender_id
receiver_id







					КПІ.ІП-5102.045440.07.99.КЕ			
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм			
Розробив		Волков І.А.				Аркуш	Аркушів	
Перевірив		Ліщук К.І.						
Т. кон.					Веб-застосування пошуку менторів та помічників з програмування	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-51		
Н. кон.		Ліщук К.І.						
Затвердив		Ліщук К.І.						



Інв. № підл.	Підп. дата	Інв. № взам. підл.	Інв. № дубл.	Підп. дата

КПІ.ІП-5102.045440.07.99.СС

КПІ.ІП-5102.045440.07.99.СС

Зм.	Арк.	№ докум.	Підп.	Дата
Розроб.		Волков І.А.		
Перев.		Пищук К.І.		
Т. Кон.				
Н. Кон.		Пищук К.І.		
Затв.		Пищук К.І.		

Схема структурна варіантів використання

Літ.	Аркуші	Арк.	Аркушів
			1

КПІ ім.Ігоря Сікорського
Кафедра АСОІУ
гр. ІП-51